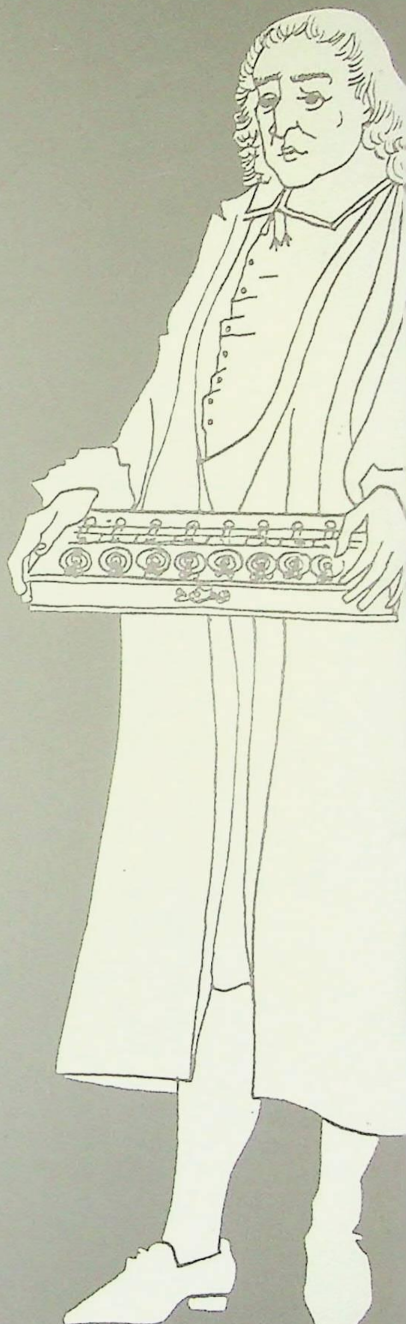
פסקל

שפה לתכנות מבני

פרופ׳ אבא אנגלברג



למד את שפת PASCAL

ערך יצחק עמיהוד

ספרי לימוד והכשרה במדעי המחשב

הוצאת ״הוד־עמי״ בשיתוף המכון לפריון העבודה והייצור

פסקל

שפה לתכנות מבני

למשה ולחולדה

בברכת חברות 6/86

[illegible]

מ ו ק ד ש

לאשתי רות

ולבניי
אהרן, יעקב, דניאל ואליהו צבי

תודתנו נתונה לחברת יבמ ישראל בע"מ, אשר איפשרה לנו באדיבותה להשתמש במדפסת האיכות Quietwriter להדפסת הספר.

שרה'לה וזוהר ראויים לתודה על טרחתם הרבה בהקלדה של כתב היד, בעבודת העריכה של הספר ובעיצובו לדפוס.

יבוא על הברכה משה ראב, אשר כתב את הנספח על טורבו-פסקל.

הסדר והעריכה של הספר נעשו במחשב IBM PC באמצעות מעבד התמלילים WordMill של חברת אינטרסופט בע"מ.

פ ס ק ל
שפה לתכנות מבני

פרופ׳ אבא אנגלברג

למד את שפת PASCAL

יצחק עמיהוד

ספרי לימוד והכשרה במדעי המחשב
הוצאת ״הוד־עמי״ בשיתוף המכון לפריון העבודה והייצור

PASCAL

A Language for Structured Programming

by
Prof. A. Engelberg
edited by **I. Amihud**

הוצאת הוד עמי
לספרי מחשבים בע"מ

ת.ד. 560, רמת-גן 52105

הודפס בישראל
אייר תשמ"ו, מאי 1986

Hod-Ami Ltd
P.O.B. 560 Ramat-Gan
Israel, May 1986

הקדמה

על השפה ועל תכנות מבני

שפת פסקל (PASCAL) קרויה על שמו של המתמטיקאי והוגה הדיעות בלייז פסקל (Blaise Pascal), אשר חי בין השנים 1623-1662 בצרפת. פסקל המציא את מכונת החישוב PASCALINE, הראשונה מסוגה, כדי לסייע לאביו בחישובי המיסים. עבודותיו המדעיות והתיאורית שלו בתחום המיחשוב תרמו רבות לפיתוח מדעי המחשב של זמננו.

שפת התכנות פסקל פותחה ע"י פרופסור ניקלוס וירת (Prof. N. Wirth) בבית הספר הפדרלי פוליטכני (ETH) בציריך, שוויצריה. תכנון השפה התחיל ב-1969 והסתיים לאחר כשנה, כאשר נכתב המהדר הראשון. הגירסה התיקנית הראשונה של שפת פסקל פורסמה בשנת 1974.

שפות התכנות הרבות שהיו קיימות בתקופה זו נחשבו כפשוטות, מורכבות, או מבוססות על שיטות שלא התאימו לעקרונות של התכנות המבני (Structured Programming). פרופ' וירת טען שהן אינן מתאימות למתחילים ולעבודה מעשית, מכיון שהן אינן עיקביות ואינן "הגיוניות" ולכן החליט לפתח שפה חדשה.

מפאת החשיבות של עקרונות התכנות המבני, אשר שפת פסקל נוהגת לפיהם, ראוי שנקדיש להם דברים אחדים:

לבעיה מסוימת אחת, אשר עומדת לפתרון במחשב, ניתן למצוא כמה וכמה דרכי פתרון אפשריות בעזרת הצגה גרפית. להצגה גרפית כזאת, אשר באה לידי ביטוי בתרשים הזרימה, ישנם מספר יתרונות:

א. ניתן להציג בצורה נוחה וברורה את הקטעים העיקריים של התכנית ואת הקשרים ביניהם.
ב. ניתן לראות את הרצף של הפעולות העיקריות בכל קטע של התכנית בנפרד, ואת רצף הקטעים העיקריים המרכיבים אותה.
ג. ניתן לקרוא ולהבין את התכנית, בלי להכנס לכל פרטיה.

במשך שנים רבות נהגו תכניתנים לפתח כללי עבודה משלהם ביצירת פתרונות לוגיים בתכניותיהם, אך ככל שהיתה התכנית גדולה - כך הלך והסתבך המבנה

והאחזקה שלה היתה יותר קשה ומורכבת. מכאן נבע הצורך להכניס סדר ושיטה בתחום זה של עיבוד נתונים.

אנשי מדע מתחום המחשבים והמתימטיקה פיתחו שיטות, אשר איפשרו ליצור תכניות בעזרת כלים וסטנדרטים קבועים, כדי שהטיפול בהן יהיה יותר קל ואחזקתן לא תתקל בקשיים. אחת מן השיטות שפותחו נוגעת לתחום התרשימים ולכתיבת תכניות, והיא נקראת תכנות מבני.

שיטת התכנות המבני מבוססת על תיאורמה מתימטית, הקובעת, שניתן לכתוב כל תכנית בעזרת שלושה סוגי מבנים בסיסיים, אשר מוגדרים מראש:

א. מבנה סדרתי (Sequence).
ב. מבנה איטרציה (Iteration).
ג. מבנה בחירה (Selection).

את המבנים השונים מציגים בתרשימים בעלי תבנית קבועה, שאין לסטות ממנה. לכל מבנה יש נקודת כניסה אחת ונקודת יציאה אחת ובמידה שדרוש, אפשר לחלק אותו בחלוקה מפורטת יותר, בהתאם לכללי התכנות המבני.

בספר זה, ההדרכה לתכנות בשפת פסקל הינה על פי עקרונות התכנות המבני, אשר מאפשרים פיתוח של תכניות בעלות מבנה פשוט וברור, גם כאשר פותרים בעיות מורכבות.

לשפה התכנות החדשה של וירת היו מספר תכונות בולטות:

א. מהדר יעיל וחסכוני שיכול לפעול גם במחשב זעיר.
ב. פשטות רבה לעומת השפות שקדמו לה: FORTRAN, COBOL, PL/I.
ג. פונקציות משופרות לעומת FORTRAN ו-BASIC.
ד. מספר רב של מבנים המתאימים לעקרונות התכנות המבני (עיין בפרקים 1 עד 4).
ה. יישום פשוט של מספר מבני נתונים: רשומות, רשימות משורשרות וקבוצות (עיין בפרקים 10-14, 16, 17).
ו. תמיכה בשיטות תכנות מיוחדות, כגון רקורסיה ומשתנים מצביעים (ראה פרקים 15, 17).

התכונות ד' עד ו' מאפיינות גם את השפה PL/1, שפותחה ע"י יבמ. שפה זו מורכבת ועשירה יותר משפת פסקל וכוללת תכונות נוספות, אשר דרושות בתחום עיבוד הנתונים המינהלי ובתחום המדעי/הנדסי. בשפת פסקל הדגש הוא על פונקציות השירות הבסיסיות האופייניות לעיבודים מדעיים/הנדסיים, על משמעת התכנות המבני ועל הגדרות מראש ללא ברירות מחדל (Default Options). התיכון (Design) הזה של שפת פסקל תרם לכך שקל יהיה להתאים אותה לפעולה אף במחשבים קטנים וזעירים.

הגירסאות הראשונות של שפת פסקל פותחו תחילה עבור מחשבים גדולים. כאשר הגיעו המיקרו-מחשבים לשוק פותחו מהדרים גם בעבורם ומטבע הדברים נוצרו גירסאות שונות של השפה, כשם שנעשה גם בשפות תכנות אחרות. ההבדלים ביניהן אינם משמעותיים ורובם הינם בתחום הטיפול ביחידות הקלט והפלט בגלל השוני בתכונות של היחידות האלו, המשווקות ע"י יצרנים שונים.

במיקרו-מחשבים ובמחשבים אישיים נפוצה בשימוש גירסת טורבו-פסקל (Turbo

(Pascal). היא מאופיינת בהתאמתה לקיבולת ולעוצמה של המחשב הקטן ונותנת מידה רבה של נוחות תפעול בעזרת תפריטי ההנחייה והעורך (Editor), אשר משולבים עם המהדר. על גירסת טורבו-פסקל ועל ההבדלים בינה לבין שפת פסקל התיקנית אפשר לקרוא בנספח ב' של ספר זה.

על הספר

הספר שלפנינו עוסק בנושאים שמהווים את הקורס ללימוד שפת פסקל ברמה תיכונית ואוניברסיטאית, לפי הדרישות של משרד החינוך והתרבות והמכון להכשרה טכנולוגית (מה"ט). הוא מתאים ללימוד בכיתה וגם ללימוד עצמי.

הספר מציג בהדרגה את הבסיס לתכנות בשפה זו ועובר לנושאים מורכבים ולתכונות המיוחדות אותה. המתחיל בלימוד תכנות ייטיב לעשות אם יעיין בנספח א' ובספרות המבוא לעיבוד נתונים, לפני שהוא ניגש ללימוד השפה. הסעיפים והשאלות, אשר מיועדים ללימוד מתקדם סומנו בכוכבית (*) ואפשר לדלג עליהם מבלי לפגוע ברצף הלימוד.

הספר מדריך את הלומד להכיר את מרכיבי השפה ובעזרת הסברים ודוגמאות הוא מנחה אותו לעצב תוכנית; להשתמש במבנים פשוטים ומורכבים, בפונקציות ובפרוצדורות; לטפל בנתונים, במערכים ובמבנים של רשומות; לערוך תמליל; להפעיל שיטות מיון וחיפוש; לנצל שיטות עיבוד מתקדמות ועוד. בנספח ב' ניתנת הנחייה כיצד להשתמש גם בטורבו-פסקל ומוסברים ההבדלים בין גירסה זו של השפה לבין הגירסה התיקנית.

המתודולוגיה היא הצגת נושא ותרגול מיידי שלו, ע"י הדגמה של תוכנית בצירוף הסברים מפורטים. בסוף כל פרק ניתנו תרגילים רבים, אשר כוללים שאלות הבנה וחזרה ושאלות תכנות.

<u>שמונת הפרקים הראשונים</u> הם הבסיס ללימוד השפה ולשאר פרקי הספר.

<u>בפרק 1</u> מוצגות בפני הסטודנט תוכניות פשוטות המורכבות מקלט של נתונים, פעולות חישוב פשוטות ופלט של תוצאות. בפרק זה ועד פרק 5 עוסקים במשתנים המייצגים ערכים מספריים בלבד.

<u>בפרקים 2, 3, 4</u> מוצגים מבני הבקרה המאפיינים תכנות מבני בכלל ותכנות בפסקל בפרט.

<u>בפרק 5</u> עוסקים במערכים, שהם מבני הנתונים הבסיסיים ביותר.

<u>בפרק 6</u> מוצג, מוסבר ומתורגל השימוש בנתונים לא נומריים, דהיינו תווים ומחרוזות.

<u>בפרק 7</u> מוצגות שיטות שונות של חיפוש, מיון ומיזוג, שאפשר להפעיל גם על נתונים נומריים וגם על מחרוזות.

<u>בפרק 8</u> עוסקים בשלב נוסף של תכנות מבני באמצעות פרוצדורות ופונקציות.

<u>הפרקים (9)-(17)</u> שייכים לשלב השני של לימוד השפה. מומלץ ללמוד אותם באופן מסודר, אך אפשר לדלג על נושאים לפי העניין. בפרקים אלה ישנם נושאי תכנות,

אשר חלקים מהם חורגים ממסגרת הלימוד של התלמיד המתחיל. לנוחותו, סומנו בכוכבית (*) סעיפים ושאלות, אשר שייכים לשלב הלימוד המתקדם. מן הראוי שהלומד יהיה בקיא בשיטות אחזקת המספרים במחשב, כפי שמוסבר בנספח א'.

פרק 9 מטפל בכמה נושאים מתקדמים העוסקים בפרוצדורות ואפשר לדלג על כולו, או על חלקים ממנו, אם הנושאים אינם מתאימים למסגרת הלימודים.

פרק 10 דן בנושא של רשומות. זהו נושא מרכזי וחשוב להמשך הלימוד ועל כן מומלץ לשקוד על הדוגמאות לשימוש ברשומות.

פרקים 11, 12, 13 מהווים יחידה אחת המטפלת בנושא קבצים. אפשר לדלג על נושאים מסוימים אם אין בהם עניין מיוחד כמו למשל, פרוצדורות שונות לעיבוד ומיון קבצים, אשר בדרך כלל מסופקות בתוכנית שרות ולא נכתבות ע"י המתכנת. הנושאים של קבצים ישירים ואינדקס סדרתיים אינם כלולים בכל הגירסאות של שפת פסקל ועל-כן הקורא צריך לבדוק איך ואם להשתמש בהם במהדר שברשותו.

פרק 14 עוסק בקבוצות. הוא עומד בפני עצמו ואינו מתבסס על הלימוד הקודם של קבצים.

פרקים 15, 16, 17 מהווים יחידה אחת, אם כי אפשר לדלג על פרק 15 (רקורסיה) ועל נושאים בפרק 17 העוסקים בעצים.

נספח א' מכוון לתלמידים, אשר עושים את צעדיהם הראשונים בתכנות, או שאינם מכירים שפת תכנות מדעי. הנושאים החשובים הם השיטות השונות להצגת מספרים. בכמה מפרקי הספר יש התייחסות לנושאים אלה.

נספח ב' מציג את גירסת הטורבו-פסקל ואת המהדר המיוחד לה עם תכונותיו המותאמות לנוחות המשתמש במחשב האישי ובמיקרו.

הביבליוגרפיה שבסוף הספר תאפשר לקורא והלומד המעוניין להרחיב ולהעמיק את ידיעותיו בשפת התכנות פסקל. היא מחולקת למספר חלקים, לנוחות הקורא: ספרים בעברית וספרים באנגלית וספרות רקע למבוא לענ"א ולמבנה המחשב.

מילון המונחים ואינדקס יקלו על הקורא למצוא נושאים הדרושים לו ולהכיר מונחים נפוצים בעברית ובאנגלית.

איך תלמד/י בספר?
מוצע ללמוד כל פרק ב-3 שלבים:

א. ללמוד את תוכנו של הפרק. אם יש קושי כלשהו, אפשר להריץ במחשב תכנית אחת או יותר מן הדוגמאות המופיעות בו.

ב. לענות בכתב על השאלות כדי להעמיק את הבנת החומר.

ג. להשיב על כמה מתרגילי התכנות ולהריץ אותם במחשב.

הקורא ייטיב לעשות אם יעיין בספר הוראות ההפעלה של שפת פסקל במחשב שעומד לרשותו, לפני שהוא מתחיל בתכנות.

תוכן העניינים

פרק 5 - מערכים

פרק 6 - נתונים לא מספריים

פרק 7 - חיפוש, מיון ומיזוג

פרק 8 - פרוצדורות ופונקציות

פרק 9 - יישומים של פרוצדורות ופונקציות

פרק 10 - רשומות

פרק 11 - קבצים

פרק 12 - עדכון קבצים

פרק 13 - קבצי תמליל

פרק 14 - קבוצות

פרק 15 - רקורסיה

פרק 16 - מבני נתונים

פרק 17 - מבני נתונים דינמיים

נספח א - המחשב ושיטות ספירה

נספח ב - טורבו-פסקל

פרק 1

תכניות חישוב והדפסה

1.1 פתרון בעיה בשפת תכנות

1.1.1 תכנון הפתרון

תהליך הפתרון של בעיה בעזרת המחשב כולל שני שלבים:

א. תיכון (Design).
בשלב התיכון, מתארים את הצעדים הלוגיים לפתרון הבעיה. לשם כך משתמשים בתרשימים, כפי שנראה בהמשך.

ב. תכנות (Programming), או קידוד (Coding).
בשלב התכנות מתרגמים צעדים אלה לפקודות מחשב אשר נכתבות בשפת תכנות.

סדרת הפקודות, אשר נכנה אותן תכנית (Program), גורמות למחשב לבצע פעולות מסוימות שתכליתן להגיע לתוצאה מוגדרת, או לפתרון של בעיה מוגדרת. לכאורה, צריך לכתוב את הפקודות בשפת מכונה (Machine Language) כדי שהמחשב, או רכיביו האלקטרוניים יוכלו לבצען. דרך פעולה זו איננה רצויה ולא נדון בכך כאן. במקום זה נתכנת בשפה עילית (High Level Language) - בשפת פסקל (PASCAL), אשר דומה לשפת אנוש ונוחה לתכנות. כדי שהמחשב יוכל לפעול על פי התכנית הזו צריך להסב אותה לשפת מכונה. משימה זו נעשית ללא מעורבות המתכנת, באמצעות תכנית מהדר (Compiler). לאחר ההידור מקבלים תכנית בשפת מכונה ואפשר להריץ (Run), או לבצע (Execute) אותה במחשב.

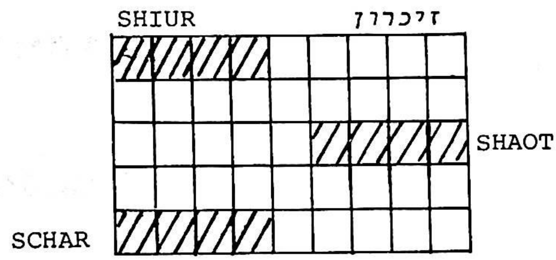
הבעיה לדוגמא שעלינו לפתור היא חישוב השכר השבועי של פועל על פי שעות העבודה שלו. את הפתרון נציג, כאמור, בתכנית מחשב בשפת פסקל. בזמן הכנת התכנית לא ידועים שיעור שכרו לשעה ומספר השעות שהוא עבד. נסתכל איפוא, על הכמויות האלו כנעלמים (או משתנים - Variables, כי ערכם יכול להיות שונה בכל הרצה של התכנית) וניתן להם את השמות SHIUR ו-SHAOT בהתאמה.

בזמן הרצת התכנית יש לספק לה ערכים מספריים עבור המשתנים SHIUR ו-SHAOT. התשובה שמחפשים, כלומר שכרו השבועי של הפועל, היא התוצאה שימסור לנו המחשב בסוף הביצוע המוצלח של התכנית. עלינו מוטל רק תפקיד אחד - לציין

מקום בזיכרון המחשב שבו תוחסן (Store), בטעות מקובל היה לומר "תאוחסן") התשובה הנכונה כאשר תהיה מוכנה. למקום זה נקרא SCHAR, שהוא המשתנה השלישי בתכנית.

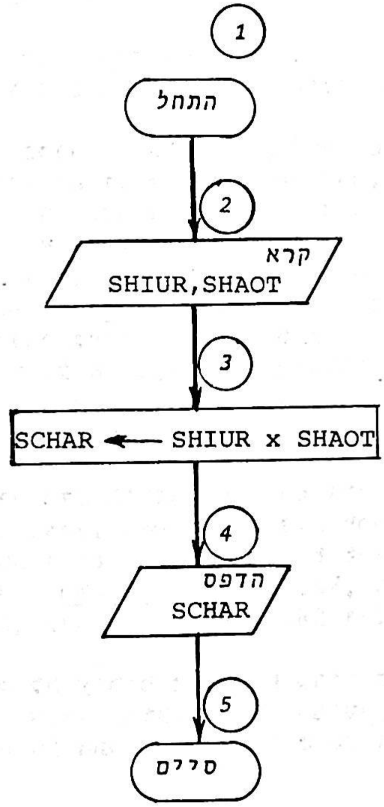
מבחינת המחשב, הנעלמים שהגדרנו מהווים שמות של תאים בזיכרון (Memory), שבהם ניתן להחזיק תווים, אשר יכולים להרכיב מספרים, אותיות או סימנים מיוחדים. כל שם מתייחס לאותו מקום בזיכרון במהלך ביצוע התכנית ולכן נקרא לו כתובת (Address), או מען.

המשתנים יכולים להופיע כך בתוך הזיכרון:



1.1.2 תרשים זרימה לפתרון בעיה

כדי לתכנת את הפתרון נכין תחילה תרשים זרימה (Flow Chart).



הסבר:

הצורות הגרפיות הממוספרות נקראות סמלים, או בלוקים (BLOCKS). המספרים ליד כל סמל מיועדים להסבר בלבד ואינם מהווים חלק אינטגרלי של התרשים.

(1) סמל זה מציין התחלה או סיום של תכנית.

(2) מקבילית מסמלת קלט (Input) - קליטה של נתונים מחוץ למחשב, או פלט (Output) - פלט של נתונים מזיכרון המחשב החוצה, למשל ע"י הדפסת התוצאות על נייר או הצגה במסוף. בבעיה שלפנינו אנו מבקשים שבזמן ביצוע התכנית המחשב יהיה מוכן לקבל מספרים שנספק ממסוף למשל. המספר הראשון מייצג את שיעור שכרו של הפועל לשעה וירשם בכתובת הזיכרון שקראנו לה SHIUR. המספר השני מייצג את מספר השעות שהפועל עבד וירשם בכתובת הזיכרון שנקראת SHAOT.

(3) מלבן מסמל תהליך, אשר יכול להיות פעולת חישוב (Calculation) או השמה (Assignment). המכפלה של השכר לשעה במספר השעות שהפועל עבד נותנת את המשכורת המגיעה לו. רוצים שהמחשב יחסין מספר זה בכתובת שנקראת SCHAR, לאחר שיסיים את החישובים.

(4) לתוצאת החישוב יש משמעות רק אם אפשר להשתמש בה. לשם כך צריך לקבל אותה על גבי המסך, או בדפוס על נייר. פעולת פלט אחרת יכולה להיות כתיבה על תקליט מגנטי, תקליטון, או סרט מגנטי. המקבילית מסמלת פעולת פלט, כפי שהוסבר קודם.

(5) כמו ב-(1).

סדר הביצוע של הפעולות הוא כסדר החיצים, ולכן מבנה כזה נקרא מבנה סידרתי (Sequential Structure). כלומר, מתקדמים מן ההתחלה עד לסיום בלי לדלג על פעולה כלשהי ומבלי לחזור על פעולה כלשהי.

1.2 תכנות בשפת פסקל

נכתוב כעת בתכנית בשפת פסקל את פתרון הבעיה שהוצגה בסעיף 1.1:

```
(1)  PROGRAM MASKORET1 (INPUT,OUTPUT);
(2)  VAR
(3)      SHIUR, SHAOT, SCHAR: REAL;
(4)  BEGIN
(5)      READ (SHIUR, SHAOT);
(6)      SCHAR:= SHIUR * SHAOT;
(7)      WRITELN (SCHAR)
(8)  END.
```

אם הקלט שהוכן למחשב נראה כך: 10.0   18.0

אז הפלט יראה כך: 1.80000E+02 כלומר: 180 (מאה ושמונים).

המספרים בסוגריים משמאל לשורות התכנית מיועדים להסבר, או בכדי לעקוב אחר הסדר הכרונולוגי של הפקודות. הם אינם חלק אינטגרלי של התכנית.
הקו מתחת למלים מסוימות מדגיש שהן נקבעות ע"י חוקי השפה וצריכות להופיע בכל תכנית בדיוק כפי שהן כתובות למעלה. אלו הן מלים שמורות (Reserved Words), אשר מרכיבות את שפת התכנות. הקו להדגשה ניתן כדוגמא בלבד.

בסעיפים הבאים נציג הסבר לתכנית. המספרים בסוגריים בראשי הפסקאות מתייחסים לשורות המסומנות של התכנית. כך ננהג גם בשאר ההסברים בהמשך הספר.

1.2.1 כותרת התכנית

(1) השם של התכנית (PROGRAM) הוא MASKORET1. שם זה ניתן ע"י המתכנת בכדי לציין את מטרת התכנית וכדי להבדילה מתכניות אחרות, אשר שמורות במחשב.
INPUT - מציין שהתכנית תצפה לנתונים מבחוץ. נתוני קלט יסופקו באמצעות התקן (Device) הקלט הסטנדרטי באותו מחשב, במקרה שלפנינו זוהי מקלדת מסוף.
OUTPUT - מציין שהתכנית תפיק תוצאות, הנקראות פלט, באמצעות ההתקן הסטנדרטי לפלט באותו מחשב, בד"כ מדפסת או מסך של מסוף.

בכדי להפריד בין משפט למשפט משתמשים בנקודה-פסיק (;).

1.2.2 הצהרת משתנים

(2) VAR מלשון VARiable (משתנה) הוא כותרת לרשימת המשתנים שישתתפו בתכנית. אסור להשתמש במשך התכנית במשתנה שלא מצהירים עליו. משפט VAR הוא משפט הצהרה (Declaration Statement) כי הוא נותן ידיעות כלליות על המשתנים בתכנית ואיננו מחייב ביצוע של פעולה כלשהי.

(3) השמות SHAOT, SHIUR ו-SCHAR נבחרו ע"י המתכנת כדי לציין את משמעותם.

שמות המסופקים ע"י המתכנת (עבור שם התכנית ועבור שם של משתנה) נבחרים לפי הכללים הבאים:

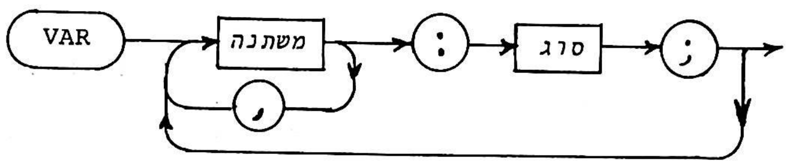
א. שם משתנה חייב להתחיל באות (ז"א אחת מן האותיות A עד Z).
ב. שאר התווים הכלולים בשם המשתנה חייבים להיות אותיות או ספרות ואסור להשתמש בכל תו אחר (למשל +,-,*,?,/).
יש מהדרים שבהם אפשר לכלול את התו _ (Underscore) בשם משתנה, ובהמשך ננצל זאת. תו זה משפר את הקריאות של משתנים כאשר רוצים ששמם יהיה מורכב משתי מלים, למשל: YELED_GADOL במקום YELEDGADOL.
ג. לא נקבע אורך מקסימלי לשם, אבל המהדר מתיחס רק ל-8 התווים הראשונים שלו. על כן, יש להמנע משמות שבהם שמונה התווים הראשונים זהים.
ד. אסור לבחור שם שהוא מלה שמורה (בסוף הפרק מופיעה הרשימה של כל המלים השמורות בשפת פסקל). דוגמאות:

שמות מותרים: TOZAAH ,A500 ,ABC ,INVENTORY
שמות אסורים: 1A50 (מתחיל בספרה)
TOZA-AH (כולל תו אסור)
PROGRAM (מלה שמורה)

אחרי הנקודתיים (בשורה 3 בתכנית) מופיעה המלה השמורה REAL, שתרגומה: ממשי. היא מגדירה משתנה שמייצג מספר ממשי, כי המשתנים שנשתמש בהם בדוגמא זו מייצגים מספרים עשרוניים (בעלי שבר עשרוני). בנוסף למשתנים מן הסוג REAL, אפשר להשתמש במשתנים מן הסוג INTEGER, ז"א משתנים אשר מייצגים מספרים שלמים.

המלה REAL מציינת שהייצוג הפנימי של המספר במחשב יהיה בשיטת הנקודה הצפה, כאשר המלה INTEGER מציינת שהייצוג הפנימי הוא במשלים לשניים (הסבר בעניין זה ראה בנספח).

אפשר לתאר את המשפט VAR בצורה סכימטית, כדלהלן:



בהמשך נשתמש בתרשים מן הסוג הזה בכדי לתאר בצורה סכימטית משפטים בפסקל. כל תו או מלה שנמצאים בתוך צורה מעוגלת חייבים להופיע ככתוב. מלה בעברית, אשר מופיעה בתוך מלבן יש להחליף במשתנה אשר נקבע ע"י המתכנת. כאשר החצים מסתעפים, פירוש הדבר שיש יותר מאפשרות אחת להתקדם והמתכנת יבחר בכיוון הרצוי בהתאם לתנאי הבעיה. מכאן, שבהצהרת VAR אפשר לרשום משתנים אחדים מכל סוג, כאשר פסיק בלבד מפריד בין אחד לחברו. אפשר גם לרשום כמה סוגים של משתנים, כאשר התו נקודה-פסיק (;) מפריד בין סוג לסוג.

1.2.3 החלק הביצועי

(4) המלה BEGIN (התחל) מציינת את תחילת הוראות הביצוע של התכנית. המשפטים לפני מלה זו הם הצהרות (Declarations). המשפטים שלאחריה הם פקודות (Instructions). המלה BEGIN נחשבת סימן פיסוק ולא משפט ולכן אין לרשום אחריה נקודה-פסיק.

קריאת נתונים

(5) הפקודה READ מציינת שמבקשים לקבל כקלט את הערכים של שני המשתנים SHIUR ו-SHAOT. נקיש במסוף שני מספרים עשרוניים המופרדים ע"י פסיק או רווח אחד לפחות. יש לזכור, שלאחר הקשת הנתונים חובה להקיש בקליד המורה למחשב לקלוט את הנתונים, בכדי שהם יועברו מהמסוף לזיכרון המחשב. קליד זה קרוי RETURN, ENTER או כדומה.

מספר ממשי אפשר לייצג בשתי דרכים:

א. ייצוג עשרוני (לקרוא משמאל לימין):

[לפחות ספרה אחת] [.] [לפחות ספרה אחת] [סימן + או -,אופציונלי]

לדוגמא, מספרים עשרוניים חוקיים: -29.4
+0.14
17.1

מספרים לא חוקיים: .017 (אין ספרה בצד שמאל)
17. (אין ספרה בצד ימין)

<u>ב. ייצוג מדעי</u> (לקרוא משמאל לימין החל מהשורה הראשונה):

[לפחות ספרה אחת] [סימן + או -,אופציונלי]
[אין נקודה, או שיש נקודה עם ספרה אחת לפחות מימין]
[ספרה או שתי ספרות] [סימן + או -,אופציונלי] [E]

התו E מחליף את המלים "בחזקת 10" כך, שהמשמעות של 5E2 היא 5 כפול 10 בריבוע.

לדוגמא, ייצוג חוקי: -26.4E7
26.4E+7
+26.4E+07
26E-7

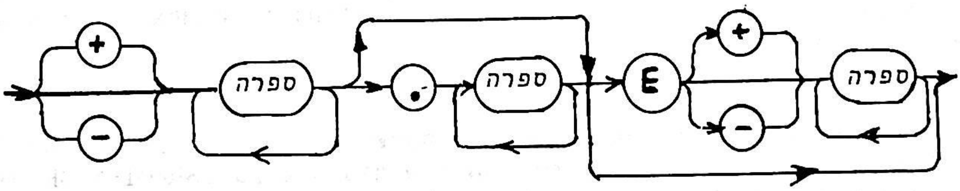
ייצוג לא חוקי: -7.5E4.1 (אחרי E יבוא רק מספר שלם)
E14 (לפחות ספרה אחת צריכה לבוא לפני E)
16.1E125 (מקסימום של 2 ספרות אחרי E)

לכן, בזמן ביצוע התכנית, אם נקיש במסוף

1.4E1 42.5

המשתנה SHIUR יקבל את הערך 14 והמשתנה SHAOT יקבל את הערך 42.5. הערכים נקראים משמאל לימין.

אפשר לסכם את הייצוג של מספרים ממשיים על ידי התרשים שמופיע למטה:



אפשר, אך אין זו חובה, לכתוב סימן (+ או -) משמאל למספר. רק האלמנטים שהם על הקו המרכזי הם חובה וגם אז, רק אם אין קו עוקף.אחרי הסימן כותבים לפחות ספרה אחת. החץ המסתעף למטה מן הספרה מראה שאפשר לכתוב מספר כלשהו של ספרות. בשלב זה אפשר לסיים את המספר הממשי, אם עוברים בחץ העוקף למעלה מן הנקודה העשרונית וכשמגיעים לקו המרכזי עוקפים את ה- E בקו התחתון. אם כוללים את הנקודה העשרונית במספר, חייבים לכלול לפחות ספרה אחת בצד ימין והחץ למטה מראה שאפשר להוסיף כמה ספרות שדרושות. עכשיו אפשר שוב פעם להפסיק, אבל אפשר גם להמשיך עם E, + או - אופציונלי ולפחות ספרה אחת, אך לא יותר משתיים, אף שלא ניתן להסיק זאת מן התרשים.

<u>מספר שלם</u> מוגדר כך (משמאל לימין):

[לפחות ספרה אחת] [סימן + או -, אופציונלי]

לדוגמא, ייצוג חוקי: -3
03
4287
MAXINT (המספר השלם הגדול ביותר שהמחשב יכול להחזיק)

ייצוג לא חוקי: 4,861 (פסיק אסור)
7- (סימן מצד ימין)

בדוגמא של חישוב השכר הקשנו 10.0 18.0
אך יכולנו גם להקיש 10 18
ולקבל תוצאה נכונה, מכיון שהשלמים הם בכלל המספרים הממשיים, כמו שראינו מן התיאור הסכימטי של מספר ממשי.

מספרים שלמים וממשיים יכולים להופיע בגוף התכנית גם-כן, ולא רק כקלט מן החוץ. כאשר הם כתובים בתכנית נכנה אותם <u>קבועים</u> (Constants).

<u>הוראות השמה</u>

(6) הוראה עם הסימן =: היא <u>הוראת השמה</u> (Assignment Statement). הסימן * מסמל פעולת כפל. כוונת ההוראה היא שהמספר שנמצא בכתובת SHAOT יוכפל במספר שנמצא בכתובת SHIUR והתוצאה תועבר לכתובת SCHAR. אם היה ערך מספרי בכתובת SCHAR לפני ההעברה, הוא <u>ימחק ויוחלף</u> ע"י המכפלה של SHAOT ב-SHIUR. משמעות הסימן =: היא כמשמעות החץ --> (ז"א מוחלף ע"י) ולא כמשמעות סימן השיוויון (=). לכן נוכל לכתוב N:=N+1, כאשר כוונתנו לקחת את הערך הקודם של N (למשל 20), להוסיף לו 1 ולהכניס את הסכום 21, שהוא הערך החדש של N, לכתובת של N. בפעולה זו נמחק את הערך הקודם (20) של N. מאידך, N=N+1 מהווה סתירה מתמטית.

פעולות אחרות, בנוסף לכפל, אשר מוגדרות בין קבועים ונעלמים הן חיבור (+), חיסור (-) וחילוק (/).

אם A ו- B הם משתנים שלמים אפשר להגדיר שני סוגים של חילוק:

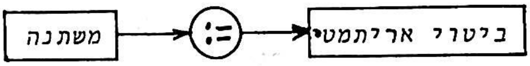
א. <u>חילוק שלם</u>: כאשר מחלקים A ב-B מקבלים שתי תוצאות של מספרים שלמים: <u>מנה</u> (Quotient) ו-<u>שארית</u> (Remainder). את המנה מקבלים ע"י הפעולה

A DIV B ואת השארית - ע"י הפעולה A MOD B. למשל, אם נתון A=10, B=3, מקבלים אחרי ביצוע הפקודה C:=A DIV B והפקודה D:=A MOD B את התוצאות C=3, D=1, בהתאמה.

ב. חילוק עשרוני: מסומן ע"י A/B. על פי הערכים שבדוגמא הקודמת, אחרי ביצוע C:=A/B נקבל C=3.333. אף על פי שהאופרנדים A ו-B הם שלמים, התוצאה היא ממשית והמשתנה C צריך להיות מוגדר כמשתנה ממשי.

בהוראת ההשמה שנמצאת בשורה השלישית (SCHAR:= SHIUR * SHAOT), כל שלושת המשתנים מוגדרים כמשתנים ממשיים. היה אסור לכתוב הוראה זו אם SCHAR היה מוגדר כמשתנה שלם, שהרי משתנה המיועד להחזיק מספרים שלמים איננו יכול להחזיק תוצאה שהיא ברוב המקרים לא מספר שלם. מאידך, אם קורה מקרה הפוך, ז"א אם SHIUR ו-SHAOT היו מוגדרים כמשתנים שלמים ו-SCHAR היה מוגדר כמשתנה ממשי, היה מותר לכתוב SCHAR:= SHIUR * SHAOT. הטיעון הקודם קביל כי אם SCHAR יכול להחזיק מספר ממשי, הוא בוודאי יכול להחזיק מספר שלם.

תיאור סכמטי של פקודת הצבה:



ביטוי אריתמטי (Arithmetic Exspresion) כולל משתנה אחד או קבוע אחד או שילוב כלשהוא של משתנים וקבועים, אשר מחוברים ע"י האופרטורים האריתמטיים (Arithmetic Operation) +, -, *, /, DIV, או MOD.

נרשום למשל כמה ביטויים אריתמטיים:

-5+7	A
10 MOD 3*4	B*A-17
	17
	15+15/5

האם הערך של הביטוי האחרון הוא 18 (15+15/5=18) או 6 ((15+15)/5=6)?
כל הספרות חייבות להיות באותה שורה במסוף, ועל כן קשה להבחין בסדר הפעולות הנכון. לשם כך קבעו חוקי קדימות לביצוע פעולות, לפי הסדר הזה:

(1) ביטוי בתוך סוגריים,
(2) סימן - , כאשר הוא מסמל שלילה ולא חיסור,
(3) MOD , DIV , / , * ,
(4) - , + .

כאשר יש אופרטורים באותה רמה, מבצעים את הפעולות משמאל לימין.

בביטוי 15+15/5 מחשבים תחילה את 15/5, שהרי חילוק (ברמה 3) מקדים את החיבור (ברמה 4) ונקבל שערך הביטוי הוא 18. אבל אם נכתוב (15+15)/5, יש לחשב תחילה את הביטוי שבתוך הסוגריים, מכיון שהוא ברמה 1 ונקבל שערך הביטוי הוא 6.

בביטוי -5+7 מחשבים תחילה -5, כי המינוס הזה מבטא שלילה (רמה 2) ואח"כ

מחברים 7 (רמה 4) ומקבלים 2. לו סדר הפעולות היה הפוך (ז"א חיבור קודם לסימן שלילה), היינו מקבלים 12-. כדי לבצע חיבור לפני שלילה כותבים את המספרים בסוגריים בצורה זו: (5+7)-.

הביטוי 4*3 MOD 10 מכיל שני אופרטורים באותה רמה ולכן צריך להתיר אותו משמאל לימין. ערך הביטוי הוא 4, כלומר 3 MOD 10 כפול 4.

כאשר הערך של ביטוי אריתמטי הוא <u>מספר שלם</u>, המשתנה שמופיע בצד שמאל יכול להיות מוגדר כשלם, רק כאשר מתקיימים שני תנאים:

א. הביטוי כולל משתנים וקבועים שלמים בלבד.

ב. הביטוי לא כולל חילוק עשרוני (כלומר /).

הביטוי 2/(A+B) למשל, נחשב ממשי בכל מקרה כי כלולה בו פעולת חילוק. כדי שהביטוי A+B יחשב ממשי, מספיק שלפחות אחד מהמשתנים A או B יהיה ממשי.

<u>הפקת הפלט</u>

(7) כדי להפיק מן המחשב את התוצאה משתמשים בפקודה WRITELN. היא מורה למחשב לכתוב, במסוף או במדפסת, את הערך של המשתנה שמופיע בסוגריים (SCHAR) ואח"כ לדלג לשורה הבאה.

<u>סיום התכנית</u>

(8) המלה <u>END</u> מציינת את סוף הקטע של הוראות הביצוע של התכנית. היות והיא לא נחשבת פקודה בפני עצמה אלא רק סימן פיסוק, היא לא צריכה להיות מופרדת מן הפקודה הקודמת ע"י נקודה-פסיק. לאחר END, אשר מציינת סוף תכנית, רושמים נקודה.

כאשר המתכנת מריץ את התכנית ממסוף, הוא מקיש בזמן ביצוע התכנית את הערכים בעבור המשתנים SHAOT ו-SHIUR ומקבל את התוצאה: 1.80000E+02 . נשתמש בתו "ˆ" כדי לסמן רווח, שיוחלף ע"י הסימן מינוס (-) אם המספר שלילי. למשל, במחשב VAX נדרשים לתוצאה 12 תווים, כאשר 4 מיועדים לחזקה, 5 - לשבר עשרוני, אחד - לנקודה, אחד - לספרה משמאל לנקודה ואחד - לרווח או מינוס, לפי המקרה.

<u>1.3 תכנון הפלט ותקשורת אדם מחשב</u>

בתכנית הקודמת קיימים מספר חסרונות:

א. אם התכנית מופעלת ממסוף, איך ידע המתכנת מתי המחשב מחכה לקלט שהוא צריך להקיש?
ב. כאשר מופיעה התוצאה, היה עדיף שיודפס יחד איתה הסבר כלשהוא, כדי שבנוסף למתכנת יהיו עוד אנשים שיבינו את משמעות המספרים או התווים המוצגים.
ג. רצוי שהתוצאה לא תהיה בייצוג מדעי, אלא בצורה יותר פשוטה, אולי אפילו כמספר שלם.

ד. אין הסבר כלשהוא בתוך התכנית ביחס למטרה, לקלט ולפלט כדי שאדם אחר, גם אם הוא מתכנת, יוכל להבין אותה בלי מאמץ או הסבר נוסף.

חסרונות אלה מתוקנים בגירסה המשופרת של התכנית, שבה כלולים הסברים שונים, כפי שמוצגים בדוגמא הבאה:

```
 (1)   PROGRAM MASKORET2 (INPUT,OUTPUT);
 (2)                     (* בתכנית זו מחשבים את המשכורת של פועל *)
 (3)   CONST
 (4)        MATANA = 25;             (* תשורה לקראת חג הפסח *)
 (5)   VAR
 (6)        SHIUR,                   (* שיעור השכר לכל שעת עבודה *)
 (7)        SHAOT: REAL;             (* מספר שעות העבודה *)
 (8)   BEGIN
 (9)       WRITELN ('MAHU SHIUR HASACHAR? KAMA SHAOT HU AVAD?');
(10)       READ (SHIUR, SHAOT);
(11)       WRITELN ('MAGIA LO SCHAR SHEL^^',
(12)            SHIUR * SHAOT + MATANA:6:2, 'AVUR':6,
(13)              ROUND(SHAOT):5, '^^SHAOT AVODA.')
(14)   END.
```

הרצה של התכנית:

הצגה במסוף:

```
MAHU SHIUR HASACHAR? KAMA SHAOT HU AVAD?
```

הזנת נתונים:

```
18.0      10.6
```

הפלט:

```
MAGIA LO SCHAR SHEL^^215.80^^AVUR^^^11^^SHAOT AVODA.
```

<u>הסבר:</u>

(1) <u>שם התכנית</u> וציון שדרושים התקנים לקלט ולפלט.

(2) <u>הערות</u> (Remarks) אינן חלק מהוראות התכנית. אפשר לכתוב אותן באנגלית או בעברית, אך בכל מקרה יש לתחום אותן בסימן מתאים. כל התווים אשר נמצאים בין סוגריים מן הסוג { } נחשבים להערה. המהדר מתעלם מן ההערה ולא מנסה לתרגם אותה והיא גם לא מופיעה כחלק מן התוצאות. מאידך, בכל פעם שמבקשים העתק של התכנית, ההערה מופיעה בגוף התדפיס ומטרתה להציג לקורא הסבר על התכנית, המשתנים, או על מהלך הפתרון.

אם זוג הסימנים {..} לא נמצא על לוח המקשים, אפשר להחליפם ע"י סוגריים וכוכביות בצורה זו: (*..*). כל תו יכול להופיע בתוך ההערה להוציא התווים } או (* , כדי שהמהדר לא יחשוב שההערה תמה כאשר הוא סורק את התווים משמאל לימין. הערה לא נחשבת כפקודה ולכן אין צורך בנקודה-פסיק בכדי להפרידה מן הפקודה הבאה.

(3) <u>קבוע:</u> מותר להשתמש בקבוע בלי להצהיר עליו מראש. מומלץ להשתמש בהצהרת CONST כדי לתת לקבוע שם לפי אותם כללים המתייחסים לשמות של משתנים. כאשר עושים כך, מונעים מן השם הזה לקבל במשך התכנית כל ערך, חוץ מן הערך הנקוב. בזה שונה ההצהרה CONST מההצהרה VAR.

נותנים שם לקבוע בגלל הסיבות הבאות:

א. כאשר משתמשים בקבוע פעמים רבות במשך התכנית (ובמיוחד כאשר הוא מספר ארוך כמו למשל 3.1415926536 = π), יותר קל להגדיר אותו פעם אחת בתחילת התכנית ולהמנע מכתיבה חוזרת ומטעויות אפשריות.

ב. אם משתמשים בקבוע פעמים רבות במשך התכנית והוא עשוי להשתנות מדי פעם בין הרצה להרצה. אפשר לשנות רק את ההצהרה CONST ולא את כל האיזכורים של אותו קבוע בתוך התכנית.

ג. כדי למנוע כפל משמעויות שונות לשם, כפי שמקובל בתפיסת התכנות המבני. הגדרת שם כ-CONST כובלת אותו מלשמש מטרה אחרת.

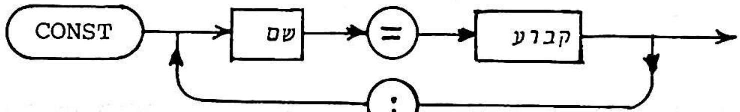
ד. אם יש חשש שמשמעות הקבוע לא תובן, השם שניתן לו עשוי לעזור. למשל, בדוגמא זו, השם MATANA מבהיר את סיבת ההוספה של הערך 25.

דוגמאות נוספות:

```
CONST    HACHNASA_MINIMALIT = 15E1;
         HACHNASA_MAXIMALIT = 15E2;
         SHIUR_MAS = 0.35;
```

שמות אלה מקבלים ערכים קבועים כפי שרשום בצד ימין של סימן השיויון ולכן נכנה אותם קבועים (Constants).

אפשר להציג את ההצהרה באופן סכימטי:



בתרשים למעלה מוצגת הצהרה של קבוע שלא ניתן לשינוי ע"י הצבה. לכן משתמשים בסימן שיויון ממש =, ולא בסימן ההשמה =: (נקודתיים וסימן שיויון צמודים). המלה CONST מופיעה רק פעם אחת, אבל מותר להגדיר קבועים אחדים, כאשר הם מופרדים זה מזה ע"י נקודה-פסיק. לשם של קבוע יש לפעמים תפקיד בהגדרת משתנה ולכן הצהרת CONST חייבת להקדים את הצהרת VAR.

(5)-(7) הצהרת משתנה: הצהרת VAR נמשכת על פני 3 שורות. אין הגבלות ביחס להמשך של פקודה או הצהרה משורה לשורה מעבר לדרישה שאסור לחלק שם של משתנה, מלה שמורה או קבוע בין שורה לשורה. כאשר כותבים הערה אפשר גם כן לגלוש משורה לשורה.

(9) פקודת כתיבה: המטרה של פקודה (9) היא להודיע למפעיל המסוף שמחכים לקלט עבור המשתנים שיוזכרו בפקודה (10), SHIUR ו-SHAOT.

בפסקל יש שתי פקודות כתיבה, WRITE ו-WRITELN. שתיהן מורות על כתיבה במסוף או על הדפסה במדפסת. בפקודה זו כותבים תמליל, אשר מוצג ע"י מחרוזת

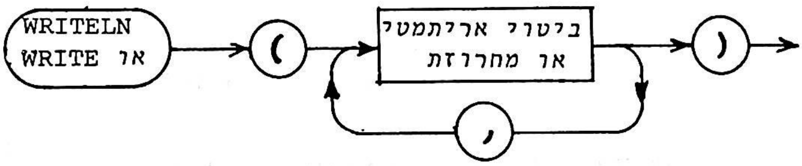
(String), שהיא שורה של תווים שמופיעים בין גרש לגרש. המחרוזת תודפס בלי שינוי, כחלק מן התוצאות ובזה היא שונה מהערה שלא מופיעה בתדפיס של תוצאות העיבוד. אסור שהמחרוזת תכיל גרש, שהרי אז המהדר יחשוב שסיימנו אותה ולכן, כאשר רוצים שיהיה גרש במחרוזת יש לכתוב שני גרשים צמודים. בפלט יודפס גרש אחד. לדוגמא,

```
WRITELN ('THIS^''^IS A GERESH')
```

פקודה זו מפיקה פלט שיראה כך:

```
THIS^'^IS A GERESH
```

הצורה הכללית של פקודת WRITE או WRITELN היא:



הפקודה WRITE או WRITELN מעבירה את הערכים המבוקשים לתוך איזור בזיכרון המחשב, אשר נקרא לו מאגר (Buffer) ונייעד אותו לבניית שורות הפלט. גודל המאגר כאורך שורת הדפסה, כלומר 132 תווים. כל שדה או נתון, אשר נעביר למאגר תופס מספר מוגדר של מקומות (תווים). מילוי המאגר מתחיל מצד שמאל (כתובת נמוכה) ומתקדם לצד ימין (כתובת גבוהה).

כאשר שורת הפלט נשלחת למדפסת, ההדפסה בפועל מתבצעת באופן אוטומטי כאשר המאגר מתמלא. לאחר ההדפסה, המאגר ממולא ע"י תווי רווח (Blank) והתהליך מתחיל מחדש. כאשר הפלט נשלח למסוף, מוצג הפלט באופן שוטף לאחר כל פקודה מבלי לחכות למילוי המאגר.

המתכנת איננו קובע את היעד הסטנדרטי של הפלט, מכיון שקביעה זו נעשית מראש ע"י האחראי לתחזוקת המערכת. ברוב המתקנים כיום, הפלט הסטנדרטי הוא למסוף. ישנה גם אפשרות לקבל תדפיס של הפלט שהופיע על המסך, או לקבל את הפלט במדפסת בלבד. השיטות לכך משתנות ממחשב למחשב ולכן לא נדון בהם כאן.

קיים הבדל בין הפקודה WRITE לבין הפקודה WRITELN. בפקודת WRITELN, אחרי הכנסת הנתון האחרון למאגר, המאגר נשלח לאמצעי הפלט גם אם הוא לא מלא. פקודת WRITE איננה מחייבת את הורקת המאגר אם הוא איננו מלא. המאגר יודפס רק אם הנתון האחרון המוכנס אליו ע"י פקודת WRITE ממלא אותו. במסוף, הפלט אמנם יופיע מיד גם אחרי שימוש בפקודת WRITE, אבל הסמן (Cursor) לא יעבור לשורה הבאה, כמו שקורה ב-WRITELN, אלא ישאר באותו מקום. פקודת WRITE הבאה תכתוב באותה שורה, מיד אחרי התווים שזה עתה נכתבו.

בדוגמא שלפנינו השתמשנו ב-WRITELN כי רצינו שהסמן יעבור לשורה הבאה לאחר הצגת השאלות. אפשר להגיע לתוצאה זו בשתי דרכים נוספות לאלו שכתובות בשורה (9):

א.

```
WRITE   ('MAHU SHIUR HASACHAR?');
WRITELN ('KAMA SHAOT HU AVAD?')
```

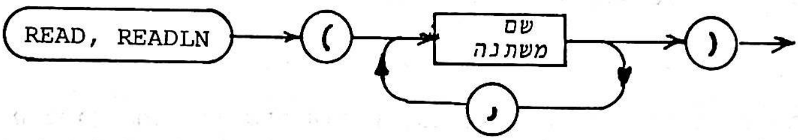
הפקודה WRITELN גורמת להמשך מילוי המאגר ולהצגת תוכנו. שני המשפטים, אשר נמצאים בשתי פקודות עוקבות, יופיעו בשורה אחת.

ב.

```
WRITE ('MAHU SHIUR HASACHAR? KAMA SHAOT HU AVAD?');
WRITELN;
```

הפקודה WRITELN גורמת להדפסה, או להצגה מיידית של תוכן המאגר, גם אם הוא לא מלא, ולמעבר לשורה הבאה. היות ו-WRITELN גורמת למדפסת או למסוף לעבור לשורה הבאה, היא יכולה להרשם בלי סוגריים עוקבים, כמו שראינו למעלה. הפקודה WRITE לעומת זה, מעבירה את הסמן רק לאחר שהיא מדפיסה את התוכן שבין הסוגריים העוקבים ולכן אין לה משמעות בלי סוגריים.

(10) פקודת קריאה: בשפת פסקל אפשר להשתמש בשתי פקודות קריאה שונות: READ ו-READLN. הצורה הכללית של פקודת READ ו-READLN היא:



הקריאה מיחידה חיצונית נעשית אל מאגר בזיכרון. הפקודה READ גורמת למצביע (Pointer) להתקדם במאגר עד שהוא נתקל במספר, ובדוגמא זו זהו המספר אשר ייקלט כערכו של SHIUR. המצביע ממשיך לחפש את הנתון עבור המשתנה SHAOT. בקריאה היזומה ע"י פקודת READ נוספת, החיפוש יתחיל מיד לאחר הנתון שהתקבל כערכו של SHAOT.

הפקודה READLN שונה מהפקודה READ. כאשר קוראים את כל המשתנים בהוראת READLN, המאגר מנוקה מתוכנו הקודם והמצביע מוזז באופן אוטומטי להתחלת המאגר, אשר מייצג כעת את שורת הקלט הבאה. פקודת READ או READLN הבאה תמלא שוב את המאגר בנתונים חדשים. היא תתחיל לטפל בנתונים החדשים שבתחילת המאגר ולא מהמקום שם היה המצביע בסיום קריאת הנתונים הקודמים. הנתונים יועברו אל המשתנים הרשומים בפקודה.

בדוגמא שלפנינו, אם במקום READ(SHIUR,SHAOT) היינו כותבים:

```
READ (SHIUR);
READ (SHAOT);
```

התוצאה היתה זהה לתוצאה הקודמת ושני הערכים היו יכולים להופיע בשורה אחת במסוף, או בשתי שורות.

נניח שכותבים כך:

```
READLN (SHIUR);
READ   (SHAOT);
```

שתי פקודות אלו גורמות לכך שאחרי קריאת SHIUR, התוכן של השורה הראשונה המוצגת במסוף היה נמחק מן המאגר והמצביע עובר לשורה חדשה. פקודות אלו מחייבות שהנתון השני לא יהיה באותה שורה במסוף.

לדוגמא, נניח שהקלט נראה כך:

השורה הראשונה במסוף:	3	2	1
השורה השנייה במסוף:	6	5	4

לאחר ביצוע הפקודות READ (A,B); READ (C,D);
ערכי המשתנים יהיו A=1, B=2, C=3, D=4

לאחר ביצוע הפקודות READLN (A,B); READLN (C,D);
ערכי המשתנים יהיו A=1, B=2, C=4, D=5

<u>קלט לתכנית</u> צריך לקיים את התנאים הבאים:

א. הנתונים צריכים להיות קבועים קבילים בפסקל, ז"א בלי פסיקים ועם ספרות משני צידי הנקודה העשרונית.

ב. כל נתון צריך להתאים למשתנה אשר הוא מוצב בו. כלומר, מספר שלם יוצב במשתנה שמוגדר כשלם ומספר ממשי - במשתנה שמוגדר כממשי. אפשר להציב מספר שלם למשתנה ממשי, אלא שהוא יוסב לצורת הייצוג של נקודה צפה.

ג. לפחות רווח אחד (או פסיק) חייב להפריד בין נתון לנתון.

ד. אסור לחלק נתון אחד בין שתי שורות במסוף, מפני שבסוף שורה המחשב מוסיף תו מיוחד שאסור לו להופיע בתוך המספר.

(11) <u>פקודת כתיבה מורכבת:</u> בפקודה זו משלבים יחד מחרוזות וביטויים אריתמטיים, אשר מופרדים ע"י פסיקים. המחשב ידפיס את הכל בסדר הרשום בפקודת WRITELN, החל מצד שמאל. בכדי שתוצאת הביטוי האריתמטי בשורה (12) לא תהיה צמודה למחרוזת שקודמת לה, הוספנו שני רווחים למחרוזת.

מהדר פסקל יעביר להדפסה את המחרוזת והביטויים שבתחום הסוגריים ברצף, החל מסוגר שמאלי ועד לסוגר ימני. כאשר מדפיסים בעברית יש קושי מסוים שעליו נעמוד בהמשך כאשר נציג דוגמא נוספת.

(12) <u>ביטוי אריתמטי בפקודת WRITE:</u> יש לשים לב לכך שבתכנית זו לא הוגדר משתנה שנקרא SCHAR. המשוואה ששימשה להגדרת SCHAR בתכנית הקודמת, נכתבה פה בפקודת WRITELN. הביטוי האריתמטי SHIUR*SHAOT+MATANA מורכב משני משתנים ממשיים וקבוע שלם - הרכב שמוביל לתוצאה ממשית.

כבר ראינו שיש <u>ברירת מחדל</u> (Default Option) למספר התווים שיוצגו במסוף, או שיודפסו. ברירת המחדל למספר ממשי בייצוג מדעי היא הקצאה של 12 מקומות במחשב VAX, או 14 מקומות במחשב יבמ. המספר 6 אחרי הנקודתיים הראשונים מציין שלא רוצים מספר באורך מלא, אלא באורך 6. המספר 2 לאחר הנקודתיים השניים מציין שלא רוצים ייצוג מדעי, אלא ייצוג עשרוני עם שתי ספרות לאחר הנקודה העשרונית.

השבר שיודפס יעוגל לשתי ספרות עשרוניות. למשל, אם התוצאה המדויקת היתה 215.8092 יודפס המספר 215.81, שהוא הערך המעוגל לשתי ספרות עשרוניות.

ברירת המחדל לאורך מחרוזת הוא כמספר התווים שבין הגרש הראשון לבין הגרש השני, אלא שגם זאת ניתן לשנות ע"י הוספת נקודתיים ומספר. למשל, המלה AVUR

תופסת רק 4 תווים, אבל ה-6 אחרי הנקודתיים יגרום לכך שיהיו שני רווחים בצד שמאל שלה. בכל מקרה שמקצים יותר מקומות מאורך המחרוזת למעשה, הם מופיעים כרווחים בצד שמאל.

(13) <u>פונקציה בפקודה:</u> SHAOT הוא משתנה ממשי שרוצים להפוך את ערכו למספר שלם, כלומר בלי נקודה עשרונית. עושים זאת בעזרת <u>פונקציה</u> (Function) שנקראת ROUND. אפשר להסתכל על פונקציה כתכנית קטנה אשר מקבלת קלט ומפיקה פלט. הקלט נקרא <u>ארגומנט</u> (Argument) והוא נרשם בין סוגריים, לאחר שם הפונקציה.

הפונקציה ROUND(X) מקבלת כארגומנט X רק מספר ממשי, לדוגמא: 7.69. התוצאה תמיד תהיה המספר השלם הקרוב ביותר לאותו מספר ממשי ובדוגמא זו - 8. מעגלים מספר עם שבר עשרוני של 0.5 בכיוון למעלה, למשל ROUND(8.5) שווה ל-9. עוד על פונקציות ראה בסעיף הבא ובפרק 8.

ברירת המחדל לגבי מספר התווים אשר מוקצים למספר שלם בזמן ההצגה במסוף או בעת הדפסה היא 10 תווים במחשב VAX ו-8 תווים במחשב יבמ. ניתן לשנות ערכים אלה ע"י הוספה של נקודתיים ואחריהם מספר. עושים זאת בפקודת WRITE או WRITELN, כאשר מציינים את המשתנה, או את הקבוע.

בדוגמא כתבנו ROUND(SHAOT):5 בכדי להודיע שנקצה 5 תווים לתוצאה השלמה של הפונקציה ROUND. אע"פ שמספר השעות לא ידרוש יותר משני תווים כתבנו 5, כדי שיהיו לפחות 3 רווחים בצד שמאל של המספר, להבדיל בינו לבין המלה הקודמת לו -'AVUR'. כדאי לציין, שאם מספר התווים המוקצים קטן מכדי להכיל את כל הספרות, מתעלמים מציון ההקצאה ומדפיסים את המספר במלואו.

<u>1.3.1 קלט ופלט של תמליל בעברית</u>

כאשר התוכן שמועבר למאגר להדפסה הוא מספרים או תמליל באנגלית, כיוון הסמן משמאל לימין הוא הכיוון הנכון. כאשר התמליל הוא בעברית האותיות נכתבות כביכול בכיוון הפוך. המחשב יתייחס לתמליל בעברית כפי שהוא מתייחס לתמליל באנגלית ויעביר את התו השמאלי לצד שמאל של המאגר, או לצד שמאל של המסך.

ישנם מסופים שבהם ניתן להציג ולכתוב אותיות בעברית אע"פ שכיוון הסמן הוא משמאל לימין, כמו באנגלית. בזמן כתיבה של תמליל בעברית במסופים אלה חייבים להקיש את האותיות בכיוון הפוך, בכיוון התנועה של הסמן. מקישים תחילה את האות השמאלית ביותר ואחריה את האות הבאה משמאל, לכיוון האות הימנית ביותר, שהיא הראשונה במשפט. אם רוצים ליישר את התמליל לימין, חייבים להוסיף רווחים משמאל, לפני ההקשה של האות הראשונה.

ישנם מסופים שבהם אפשר לעבור למצב "עברית" בצורה שבה הסמן נשאר במקומו והאותיות "נדחפות" לכיוון ימין. במסוף כזה מקישים את אותיות התמליל בעברית בצורה נכונה, מימין לשמאל. אם צריך ליישר לימין מוסיפים רווחים בסיום שורת האותיות.

קיימים מסופים שבהם תנועת הסמן אפשרית גם מימין לשמאל בנוסף לכיוון התנועה משמאל לימין. במקרה זה הזנת הנתונים פשוטה יותר ודומה להזנת נתונים באנגלית.

בספר זה לא נדון באפשרויות השונות להקשה של עברית במסוף. הקורא צריך ללמוד את תכונות המסוף שהוא משתמש בו ובמידת הצורך, להתאים את התוכניות הניתנות בספר.

1.4 פונקציות

פונקציה (Function) היא תכנית אשר ממלאת בדרך כלל משימה מוגדרת אחת. לדוגמא, יש פונקציות אשר כלולות במהדר, כמו העלאת מספר בחזקת שניים, חישוב של סינוס, שורש ריבועי, אקספוננט ולוגריתם ועוד.

הפונקציה מקבלת נתוני קלט ומפיקה כפלט ערך אחד בלבד, ששמו כשם הפונקציה. הקלט קרוי ארגומנט (Argument) ורושמים אותו בסוגריים לאחר שם הפונקציה.

הפונקציה ROUND(X) למשל, מקבלת ארגומנט ממשי ומחזירה תוצאה שלמה. פונקציה דומה לה היא TRUNC(X), שמסירה את החלק העשרוני של הארגומנט וכך הופכת אותו למספר שלם. לדוגמא: TRUNC(7.69) שווה ל-7.

יש פונקציות שמקבלות גם ארגומנט ממשי וגם ארגומנט שלם, אך התוצאה תמיד תהיה ממשית. לדוגמא,

סינוס של X רדיינים	SIN(X)
קוסינוס של X רדיינים	COS(X)
ארכטנגנס של X (ברדיינים)	ARCTAN(X)
הלוגריתם הטבעי של X	LN(X)
שורש של X	SQRT(X)
e בחזקת X	EXP(X)

יש פונקציות שהתוצאה שלהן תהיה ממשית או שלמה בהתאם לארגומנט. כלומר, ארגומנט שלם נותן תוצאה שלמה, וארגומנט ממשי נותן תוצאה ממשית. לדוגמא,

הערך המוחלט של X	ABS(X)
X בריבוע (כלומר X*X)	SQR(X)

העלאה בחזקה:

מהדר פסקל איננו מאפשר לכתוב X^Y בכדי להעלות X בחזקת Y. אם Y שווה ל-2, אפשר להשתמש בפונקציה SQR(X), אשר כלולה בו. ע"י שימוש מתוחכם בפונקציות, נוכל לבצע כל פעולה של העלאה בחזקה על פי הנוסחה הבאה:

נתון: $Z=X^Y$ ולכן: $\ln(Z)=\ln(X^Y)=Y\ln(X)$ ומכאן: $Z=e^{\ln(Z)}=e^{Y\ln(X)}$

על פי נוסחה זו נכתוב את התכנית כדי להעלות מספר בחזקה:

```
PROGRAM X_BECHEZKAT_Y (INPUT,OUTPUT);
                    (* בתכנית זו מעלים את X בחזקת Y
                       ומדפיסים את התוצאה שנקראת Z. *)
VAR
    X, Y, Z: REAL;
```

```
(6)  BEGIN
(7)    WRITELN ('עליך להזין את הערך של המספר X והחזקה Y');
(8)    READLN (X,Y);
(9)    Z:= EXP(Y*ln(X));
(10)   WRITELN ('הערך של^',X:6:2,'^בחזקת^',Y:6:2,'^שווה ל-^',Z:6:2);
(11) END.
```

תדפיס ההרצה של התכנית:

```
עליך להזין את הערך של המספר X והחזקה Y
```

קלט:

```
3.2   4.5
```

פלט:

```
הערך של^3.20^^^בחזקת^4.50^^^שווה ל-^187.57
```

<u>הסבר:</u>

(1)-(3) שם התכנית.

(4)-(5) הגדרת המשתנים שתוארו לעיל.

(7)-(8) קריאת הנתונים.

(9) לאחר קריאת המשתנים X ו-Y בשורה (8), אפשר לחשב את X בחזקת Y. <u>EXP</u> היא פונקציה שמעלה את e לחזקה שמופיעה בארגומנט. <u>ln</u> היא פונקציה שמוצאת את הלוגריתם הטבעי של הארגומנט שלה. הערך שמופיע בארגומנט של EXP הוא Y*ln(X) ולכן משווים את Z ל: $e^{Y\ln(X)}$.

(10) הדפסת הפלט. לעתים מזכירים בזמן הפלט גם את נתוני הקלט, כדי שאדם שלא הזין את הקלט אך מקבל את התוצאות יוכל להבין אותן ולהפיק מהן תועלת. מאפשרים פה רק 3 מקומות בצד שמאל של הנקודה העשרונית של Z, אבל תודפסנה כל הספרות המשמעותיות, אם יש.

<u>הפונקציות SQR(x) ו-SQRT(x)</u>

ברוב המקרים, אין צורך להשתמש בחזקות יותר גבוהות מ-2 או 3, ולא יותר קטנות מ- 1/2 (ז"א שורש). לכן נוכל להסתפק בפונקציות המוגדרות SQR להעלאה בחזקה של 2 ו-SQRT לקבלת שורש ריבועי כמו שנראה בדוגמא הבאה:

```
PROGRAM MERCHAK (INPUT,OUTPUT);
(* בתכנית זו מכניסים את הקואורדינטות
X1,Y1 ו- X2,Y2 של שתי נקודות בגרף ומחשבים את המרחק
ביניהן, שהוא הפלט. *)
VAR
  X1, Y1,          (* קואורדינטות של נקודה התחלתית *)
  X2, Y2           (* קואורדינטות של נקודה סופית *)
          :INTEGER;
  MERCHAK  : REAL;
BEGIN
  WRITELN ('הכנס נקודה התחלתית X1,Y1 ונקודה סופית X2,Y2');
  READ (X1, Y1, X2, Y2);
  MERCHAK:= SQRT (SQR(X2-X1)+SQR(Y2-Y1));
```

```
(14)     WRITELN ('(', X1:2, ',', Y1:2, ' ( המרחק בין הנקודה');
(15)     WRITELN  ('(', X2:2, ',', Y2:2, ' ( לבין הנקודה');
(16)     WRITELN  ( MERCHAK:7:2, 'ˆˆˆ:-שוה ל');
(17)   END.
```

דוגמת הרצה של התכנית:

הכנס נקודה התחלתית X1,Y1 ונקודה סופית X2,Y2
4, 3, 7, 10

דוגמת הפלט:

המרחק בין הנקודה (ˆ4,ˆ3)
לבין הנקודה (ˆ7,10)
שוה ל: ˆˆˆ7.62ˆˆˆ

<u>הסבר:</u>

(1)-(4) שם התכנית ותיאורה.

(5)-(9) הגדרת המשתנים עם הסבר מלווה.

(11)-(12) קריאת הנתונים.

(13) חישוב מרחק בעזרת הנוסחא הידועה:

$$MERCHAK = \sqrt{(X2-X1)^2 + (Y2-Y1)^2}$$

(14)-(16) הדפסת הפלט. בשורה (14) רוצים שיודפס "(ˆ4,ˆ3)". כדי לסדר זאת צריך להדפיס בנפרד 5 פרטים, משמאל לימין:

- תו ')'.
- X1 שערכו 4.
- תו ','.
- Y1 שערכו 3.
- תו '(' ותמליל מימין לו.

ל-X1 ו-Y1 קבענו רוחב של 2 במקום הרוחב הסטנדרטי של 10 מקומות לכל אחד כדי לקבל עריכה ברורה יותר של הפלט.

במקרים רבים המתכנת רוצה ליצור פונקציות חדשות, אשר מטרתן להעשיר את אוסף האופרטורים והפונקציות הקיימות של השפה בה הוא מתכנת. הן אינן תכניות בפני עצמן ואת הקלט והפלט הן מבצעות בדרך שונה. הן מקבלות ערך אחד או יותר ומחזירות ערך מחושב אחד בלבד. בפונקציות המוגדרות ע"י המתכנת נדון בהרחבה בפרקים 8 ו-9.

<u>1.5 קבועים ומשתנים שלמים</u>

<u>1.5.1 הערך MAXINT</u> (*)

המספר השלם הגדול ביותר שאפשר להשתמש בו בפסקל נקרא MAXINT. הערך שלו שונה במחשבים שונים, בהתאם לארכיטקטורה שלהם.

נתייחס לביטוי הבא: MAXINT DIV 15 * 5. על אף שהערך הסופי שלו הוא MAXINT מחולק ל-3, הוא יגרום גלישה (ראה נספח), כי המחשב לא מסוגל לטפל בו כראוי. כאשר המחשב מנסה לחשב MAXINT*5 כבר נוצרת הגלישה ואף על פי שהתוצאה הסופית תהיה קטנה מ-MAXINT. אם נכתוב 5.0*MAXINT/15 לא תהיה בעיה, כי התוצאה של הכפלת ערך ממשי (5.0) בערך שלם (MAXINT) היא ערך ממשי והמחשב יכול להחזיק מספר ממשי שערכו MAXINT*5.

המחשב מסוגל להחזיק מספרים ממשיים הרבה יותר גדולים מ-MAXINT, למשל עד 10^{38} במחשב VAX. צריך לדאוג לכך שהתוצאה של פעולה כלשהיא בשלב כלשהו לא תהיה ערך שלם גדול מ-MAXINT. למשל, הפעולה ROUND(2.0 * MAXINT) DIV 3 תגרום לגלישה. חישוב הארגומנט יתבצע כי התוצאה של 2.0*MAXINT תהיה ממשית, אך הפונקציה ROUND הופכת תוצאה זו למספר שלם אשר גדול מדי. אם נעשה חילוק עשרוני כחלק מן הארגומנט, כלומר ROUND(2.0*MAXINT/3) לא תהיה גלישה. במקרה זה הארגומנט יהיה פחות מ-MAXINT לפני שהפונקציה תהפוך אותו לערך שלם.

כאמור למעלה, ערכו של MAXINT שונה במחשבים שונים: ב-PDP-11 הוא שווה ל-32,767, שהוא המספר הגדול ביותר שאפשר להחזיק במלה של 16 סיביות, דהיינו $2^{15}-1$. במחשבים VAX ו-IBM/370 בעלי מלה של 32 סיביות הערך של MAXINT הוא 2,147,483,647 דהיינו $2^{31}-1$.

כאשר קוראים משתנים שלמים הקריאה תפסק באחד משני מקרים:

א. כאשר מגיעים לתו שלא מתאים למספר שלם. למשל אות, נקודה עשרונית או כל תו שאיננו ספרה.
ב. כאשר מזינים ספרה שגורמת לגלישה. כלומר, המספר שמנסים להכניס בקלט גדול מ-MAXINT .

כאשר קוראים לתוך משתנה ערך גדול מן הערך אשר מותר להחזיק באותו מחשב, מקבלים תוצאה בלתי צפויה, שאין להתחשב בה. האחריות לכך הינה בידי התכניתן, אשר צריך למנוע אותה. ישנם מחשבים אשר מתריעים על פעולה כזו ועל המתכנת לקלוט את ההודעה ולטפל בבעיה.

אם נקרא כקלט את הערך 15.72, המשתנה השלם A יקבל את הערך 15. הנקודה מציינת למחשב שסיימנו לקרוא את ערכו של A, שהרי נקודה לא יכולה להיות חלק ממספר שלם.

1.5.2 פעולות במשתנים שלמים

בתכניות הבאות כל המשתנים הם שלמים ולכן צריכים לוודא שערכם לא יעלה על MAXINT, לא בזמן הקריאה ולא כתוצאה מפעולות כשלהן.

דוגמא ראשונה:

בתכנית זו נניח שנתון מספר בן 3 ספרות. התכנית תדפיס אותו בסדר ספרות הפוך.

```
(1)  PROGRAM HAFOCH MISPAR (INPUT,  OUTPUT);
(2)  ( * תכנית זו הופכת את הסדר של מספר הקלט * )
```

```
(3)  VAR
(4)     MISPAR, YECHIDOT, ASAROT, MEOT: INTEGER;
(5)  BEGIN
(6)     WRITELN ('הקש מספר בן 3 ספרות');
(7)     READLN (MISPAR);
(8)     YECHIDOT := MISPAR MOD 10
(9)     ASAROT := ( MISPAR MOD 100) DIV 10 ;
(10)    MEOT := MISPAR DIV 100 ;
(11)    WRITELN ('בסדר ספרות הפוך הינו:^', MISPAR:1,'המספר^');
(12)    WRITELN  (YECHIDOT:1, ASAROT:1, MAOT:1)
(13) END.
```

כאשר נריץ את התכנית נקבל:

```
הקש מספר בן 3 ספרות
532
המספר 532 בסדר ספרות הפוך הינו:
235
```

<u>הסבר:</u>

(1)-(2) שם התכנית.

(3)-(4) רשימת המשתנים שישתתפו בתכנית.

(5) התחלה של החלק הביצועי של התכנית.

(6)-(7) קריאה של המספר שיש להפוך את סדר הספרות שלו.

(8) פעולת MOD מספקת את ספרת היחידות, שהיא השארית בחלוקת המספר ב-10.

(9) מוצאים את השארית ע"י חלוקה ב-100 ומקבלים את ספרת היחידות ואת ספרת העשרות. כדי למצוא את ספרת העשרות בלבד מחלקים ב-10 ולא מתחשבים בשארית.

(10) חילוק ב-100 ללא שמירה של השארית יתן את ספרת המאות.

(11)-(12) מקצים רוחב של תו אחד ל-MISPAR אף שיודעים שהוא יותר רחב מ-1. אם קובעים רוחב של 1, המספר יתפוס רק תו אחד, אם הוא בן ספרה אחת בלבד. המספר יוצג במלואו אם הוא כולל ספרות רבות יותר, כמו במקרה שבדוגמא.

שיטה זו מבטיחה שלא יווצרו רווחים בזמן תצוגת הפלט של MISPAR, כי הוא מוכרח לתפוס יותר ממקום אחד. כל אחת מהספרות YECHIDOT, ASAROT ו-MEOT תופס מקום אחד בלבד ולכן מקצים להן רוחב של 1 כדי שיוצגו זו ליד זו ויראו כמספר אחד בן 3 ספרות.

(13) סוף החלק הביצועי של התכנית.

דוגמא שנייה:

במרכזת טלפונים יש מכשיר חדש שמודד את זמן ההתחלה של כל שיחה ואת משך השיחה בשניות. יש לכתוב תכנית שתפיק את זמן הסיום של כל שיחה:

```
PROGRAM ZMAN_SICHA (INPUT, OUTPUT);
    (* בתכנית הזו קוראים כקלט את זמן ההתחלה של השיחה, לפי
   שעות, דקות ושניות בשעון של 24 שעות. משך השיחה ניתן בשניות.
   הפלט יהיה זמן סיום השיחה. *)
VAR
      SHAAI, DAKAI, SHNIYAI,        (* זמן התחלת השיחה *)
      SHAA, DAKA, SHNIYA,        (* זמן סיום השיחה, לחשב *)
      ORECH_HASICHA :INTEGER;       (* אורך השיחה בשניות *)
BEGIN
      WRITELN ('הקש שעה, דקה ושנייה של התחלת השיחה');
      READLN ( SHAAI, DAKAI, SHNIYAI);
      WRITELN ('הכנס את משך השיחה בשניות');
      READLN ( ORECH_HASICHA);
                              (* חשב את זמן סיום השיחה *)
      SHNIYA:= SHNIYAI + ORECH HASICHA;
      DAKA:= DAKAI + SHNIYA DIV 60;
      SHNIYA:= SHNIYA MOD 60;
      SHAA:= SHAAI + DAKA DIV 60;
      DAKA:= DAKA MOD 60;
      SHAA:= SHAA MOD 24;
      WRITELN ('משך השיחה הוא^', ORECH_HASICHA:1, '^שניות');
      WRITELN;
      WRITELN (SHAAI:2, '/', DAKAI:2, '/',
                  SHNIYAI:2, 'השיחה החלה בזמן^');
      WRITELN;
      WRITELN (SHAA:2, '/', DAKA:2, '/',
                  SHNIYA:2, 'השיחה הסתיימה בזמן^');
END.
```

דוגמא להרצת התכנית:

<u>קלט:</u> הקש שעה, דקה ושנייה של התחלת השיחה

23 56 35

הכנס את משך השיחה בשניות

3421

<u>פלט:</u> משך השיחה הוא 3421 שניות

השיחה החלה בזמן 23/56/53

השיחה הסתיימה בזמן ^0/53/36

<u>הסבר:</u>

(4)-(1) שם התכנית.

(6) משתני הקלט.

(7) משתני הפלט.

(8) משך השיחה - משתנה קלט.

(13)-(10) הכנסת הקלט.

(20)-(14) חישוב זמן סיום השיחה: מחברים את משך השיחה בשניות למספר השניות של זמן התחלת השיחה (15). הסכום יכול להיות גדול מ-60 שניות. לכן ב-(16) מחלקים את מספר השניות ב-60 בכדי לראות כמה דקות נכללות בו. מחברים את מספר הדקות שמקבלים למספר הדקות של התחלת השיחה. אחרי שהוצאנו את הכפולות של 60 בשורה (16), משתמשים בהוראת MOD שבשורה (17) בכדי להוריד את מספר השניות למספר בין 0 ל- 59: מחלקים את מספר הדקות ב-60 ומחברים את התשובה לשעת התחלת השיחה (18). לאחר שעושים זאת אפשר לראות כמה דקות בודדות נשארו (כלומר מספר קטן מ-60), שורה (19).

אבל עכשיו יש בעיה אחרת: אולי מספר השעות יחד הוא יותר מ-24, שהוא מספר השעות המקסימלי ביממה. מטפלים בבעיה זו ב-(20).

(27)-(21) הפקת הפלט. הכנת הפלט קצת מורכבת, כי בנוסף להודעה על התחלת השיחה או על סיומה, רוצים שהזמן יוצג בצורת XX/XX/XX, כלומר בצורה: שניות/דקות/שעות. לכן ב-(23) ו-(26) הפלט מורכב מ-5 צעדים:

א. הדפסת SHAAI (קלט) או SHAA (פלט), כאשר מקצים לשדה רק 2 תווים.
ב. הדפסת התו '/'.
ג. הדפסת DAKAI או DAKA, כאשר מקצים לשדה רק 2 תווים.
ד. הדפסת התו '/'.
ה. הדפסת SHNIYAI או SHNIYA, כאשר מקצים לשדה רק 2 תווים.

1.6 <u>משתנים ומספרים ממשיים</u> (*)

בתיאור הסכימטי של מספר ממשי נאמר, שמותר לרשום כמה ספרות שרוצים לפני ואחרי הנקודה העשרונית ושמותר לרשום עוד שתי ספרות אחרי האות E. מספר הספרות המירבי שאפשר לכלול לפני או אחרי הנקודה העשרונית נקבע לפי מספר הספרות המשמעותיות בייצוג עשרוני שאפשר להחזיק במנטיסה של ייצוג המספר בנקודה צפה.

במחשבים שבהם המנטיסה מורכבת מ-23 סיביות, מניחים שהיא מתחילה תמיד בספרה הבינרית 1 ואז כאילו היא מורכבת מ-24 סיביות. המספר הגדול ביותר שאפשר להחזיק ב-24 סיביות הוא $2^{24}-1$, ערכו העשרוני הוא 16,777,215 ואפשר להציג אותו ב-8 ספרות עשרוניות. כאשר יש מספרים עשרוניים בני 8 ספרות שערכם גדול יותר, לא נוכל להציג את ערכם במנטיסה זו. לכן אפשר לומר שנוכל

להחזיק בצורה משמעותית רק את טווח הערכים של מספרים בני 7 ספרות עשרוניות או פחות. מספר הספרות המשמעותיות לפני הנקודה העשרונית ואחריה לא יכול במקרה זה לעלות על 7. יש אפשרות להוסיף עוד 32 סיביות למנטיסה, ע"י "חיבור" של שתי מלים ובכך לייצג מספר בנקודה צפה במנטיסה של 56 (24+32) סיביות. מספר הספרות העשרוניות המשמעותיות שאפשר להחזיק במנטיסה עולה אז ל-16. מספרים אלה, שבהם מספר הספרות המשמעותיות גדול יותר מהרגיל הם <u>מספרים בדיוק כפול</u> (Double Precision Numbers).

האקספוננט של מספר ממשי הינו המספר שאחרי ה-E מימין. אפשר להחזיק חזקה בינרית שערכה עד 127 ומכאן שהמספר הגדול ביותר שאפשר להחזיק שווה למספר הבינרי $0.1111111*2^{127}$ שערכו העשרוני הוא $1.7*10^{38}$. לכן, המספר שאחרי ה-E יכול להיות מספר שלם עד 37 וגם 38, אם המנטיסה קטנה מ-1.7. המספר הקטן ביותר שאפשר להחזיק הוא $2.94*10^{-39}$, כאשר המנטיסה חייבת להיות 0.5 לפחות, או במחשבים אחרים $1.47*10^{-39}$. החזקה הקטנה ביותר שאפשר להחזיק בדרך זו היא 38- וגם 39-, אם המנטיסה גדולה מ-2.94, או במחשבים אחרים 1.47.

אם ההבדל בין שני מספרים מתבטא רק בספרות הלא משמעותיות, אסור להתייחס להפרש שביניהם.

לדוגמא, נתון:

	2 * 500,000.7	-	(900,000 + 100,000.6)	
מספרים מחושבים:	1,000,001.4	-	1,000,000.6	= 0.8
מספרים מעוגלים:	1,000,001	-	1,000,001	

על אף שערכו של ביטוי זה שווה ל-0.8, אם מחזיקים רק 7 ספרות ערך משמעותיות בנקודה צפה, שני המספרים יעוגלו ל-1,000,001 וההפרש שלהם יהיה שווה ל-0.

לדוגמא, נתון הביטוי הבא: 20/6 * 6 = 20

אם מחזיקים רק 7 ספרות משמעותיות, החישוב צריך להיות בשתי פעולות נפרדות:

א. 20/6 = 3.333333

ב. 3.333333 * 6 = 19.99999

1.7 המבנה של התכנית - סיכום

כפי שראינו, תכנית בפסקל מורכבת מ-3 חלקים:

א. <u>כותרת התכנית</u> (Program Heading) - שם התכנית ומכשירי קלט/פלט.
ב. <u>חלק ההצהרות</u> (Declaration Section) - תיאור של הנתונים, אשר מעבדים בחלק השלישי.
ג. <u>חלק הביצוע</u>, או <u>הפקודות</u> (Statement Section) אשר מכיל את כל הפקודות שתתבצענה בתכנית.

חלקים ב' ו-ג' יחד נקראים <u>בלוק</u> (BLOCK). מכאן, שכל <u>תכנית בפסקל</u> מורכבת <u>מכותרת ובלוק</u>. חלקים אלו מתוארים בשלד של תכנית שמופיעה למטה.

```
PROGRAM shem (INPUT ,OUTPUT);          {כותרת התכנית}          חלק ראשון:

CONST                          {חלק ההצהרות, יכול להיות ריק}     חלק שני:
        hatzhara
        ........;
                                      {המשך ההצהרות}
VAR
        hatzhara
        ........;

BEGIN                                                            חלק שלישי:
        pekuda;     {חלק הפקודות לא יכול להיות ריק}
        pekuda;
        ........;
        ........                       {אין צורך ב-;}
END.
```

בין החלק הראשון לבין החלק השני ובינו לבין החלק השלישי רושמים נקודה-פסיק. כדי להפריד בין ההצהרות השונות ובין הפקודות השונות רושמים גם-כן נקודה-פסיק. בסוף החלק השלישי רושמים נקודה.

חשוב מאוד, מנקודת ראות של הנדסת אנוש, להקפיד על הכללים לכתיבת התכנית. כתיבה נכונה תאפשר קריאה ובדיקה מהירה של התכנית ע"י המתכנת או ע"י אדם אחר. זאת היא גם אחת מהדרישות של תכנות מבני מסודר.

בשלד התכנית נרשמו המלים PROGRAM, CONST, VAR, BEGIN ו-END כשהן מיושרות בשוליים השמאליים ורשומות אחת למטה מחברתה. התוכן של כל חלק מוכנס ימינה. יש שמקפידים גם לרשום את התוכן של ההצהרות CONST ו-VAR בשורות נפרדות מן השורה בה מופיע שם ההצהרה. שיטה זו מומלצת כאשר לתוכן ההצהרה דרושה יותר משורה אחת. נוהג נוסף שתורם לקריאות של התכנית ולנוחות התיקון והאחזקה שלה הוא להגדיר כל קבוע בהצהרת CONST. הקבועים היחידים שיופיעו בתכנית ללא הצהרה יהיו המספרים 1 או 0.
האותיות ה"קטנות" שבשלד התוכנית תוחלפנה בהצהרות ובפקודות מתאימות.

מלים שמורות:

המלים המודגשות הן מלים שמורות (Reserved Words), אשר שייכות לשפת התכנות פסקל ואסור למתכנת להשתמש בהן למטרה שונה מזו שהוגדרה. בפרקים הבאים נלמד את המשמעויות של מלים נוספות לאלו שלמדנו עד עתה. רשימת המלים השמורות ניתנת להלן:

AND	ARRAY	BEGIN	CASE	CONST	DIV	DO
DOWNTO	ELSE	END	FILE	FOR	FUNCTION	GOTO
IF	IN	LABEL	MOD	NIL	NOT	OF
OR	PACKED	PROCEDURE	PROGRAM	RECORD	REPEAT	SET
THEN	TO	TYPE	UNTIL	VAR	WHILE	WITH

1.8 תרגילים

1.8.1 שאלות

(1) בהמשך נתונים השלבים לחישוב האלכסון וההיקף של מלבן באורך A ורוחב B. התבסס עליהם כדי להכין תרשים זרימה מתאים.

(א) התחל.
(ב) קרא A, B.
(ג) C שווה לשורש A בריבוע פלוס B בריבוע.
(ד) D = 2A + 2B.
(ה) הדפס C.
(ו) הדפס D.
(ז) סיים.

(2) מתוך השמות המופיעים למטה, אילו אינם חוקיים? הסבר למה.

(א) $DOLLAR
(ב) VERYLONGIDENTIFIER
(ג) TOP1
(ד) 100ACHUZ
(ה) ALPHABETIC
(ו) X-RAY
(ז) INPUT

(3) כתוב הצהרת משתנים עבור המשתנים המופיעים למטה. דייק ביחס לסוג המשתנה (שלם או ממשי).

PARTNO - מספר קטלוגי של פריט.
PRICE - המחיר של הפריט.
VAT - שיעור מע"מ.
PROFIT - הרווח לפני מס על אותו פריט.

(4) ציין לגבי כל מספר המופיע למטה אחת מתוך שלוש האפשרויות דלהלן:

- שלם.
- ממשי.
- הצגה לא חוקית (הוסף הסבר).

א. 132	ו. 2E6	יא. 10E-2
ב. 4,587	ז. 132.0	יב. 0.03E4
ג. 6.3	ח. 0.0099	יג. 1.7E999
ד. .8376	ט. 28937265	יד. 132.
ה. 4000	י. 0	טו. 3.13749

(5) נתונה הצהרת המשתנים הבאים:

```
VAR
    A, B, C : INTEGER;
    X, Y, Z : REAL;
```

אילו מתוך הפקודות הבאות שגויות:

```
(א)   C := B DIV B
(ב)   B := A MOD C
(ג)   X := B/A
(ד)   Z := X DIV C
(ה)   Y := C MOD A
(ו)   B := X + Y
(ז)   A := A + 1
(ח)   A := Z
(ט)   Z := A + 1
```

(6) כתוב את הביטויים שמופיעים למטה בצורה המקובלת בפסקל.

א. $\frac{X+2}{Y+4}$

ב. $Y + \frac{3}{Y+4}$

ג. $\frac{XY}{4} + Y$

ד. $\frac{X-4}{2 + \frac{Y}{3}}$

(7) השתמש בהצהרת CONST בכדי להצהיר על הקבועים הבאים ובחר להם שמות מתאימים.

- 453.59
- 132
- 'HI THERE'

(8) למטה רשומות 3 שורות של נתונים ושתי הצהרות של משתנים. רשום את הערכים המיוחסים למשתנים אחרי סדרת הפקודות (א), (ב) ו-(ג).

```
שורה 1:        6E2     4     -3.0
שורה 2:        3.64    25    199
שורה 3:        62      8.7   -769E4
```

```
VAR   A,B,C : REAL;
      X,Y,Z : INTEGER;
```

(א)
```
READLN (A,B,C);
```

(ב)
```
READ (A,X);
READLN (B,Y);
READ  (C);
```

(ג)
```
READLN (A,B,C);
READLN;
READLN (A,B,C);
```

(9) איך יראה הפלט לאחר הביצוע של קבוצות הפקודות הבאות (התו ^ מסמל תו רווח):

א.

```
VAR X :REAL;
:
   X:= 5;
   WRITELN ('THE^VALUE^OF^', 2+3, '^=', X);
```

ב.

```
   WRITELN ('THE^SUM^OF^2+3=', 5);
```

ג.

```
VAR A :INTEGER;
    B :REAL;
:
    A:= 1234;
    B:= -345.6789;
    WRITE (A:3);
    WRITE (B:7:2);
    WRITELN;
    WRITELN (A:6);
    WRITE  (B:5:0);
```

<u>1.8.2 תכניות</u>

(1) כתוב תכנית, אשר מקבלת כקלט משתנה בשם HEIGHT שמייצג גובהו של אדם באינצ'ים ומציגה כפלט משתנה בשם GOVA שמייצג את גובהו של אותו אדם בסנטימטרים. זכור שאינץ' אחד שווה ל- 2.54 ס"מ. הכן גם תרשים זרימה.

(2) אדם רוצה להפוך שקלים לדולרים. עבור פעולה מהסוג הזה נגבים דמי עמלה. כתוב תכנית שמקבלת את הנתונים הבאים:
- SHEKALIM - מספר השקלים ברשותו של האדם.
- ACHUZ - אחוז העמלה . בדרך כלל הוא נע בין 1/2% ל-1/8%.
- SHAAR - השער הנוכחי, ז"א מספר השקלים בדולר אחד.

הפלט הוא מספר הדולרים שהאדם יכול לקבל תמורת השקלים שהוא ממיר. הכן תחילה תרשים זרימה.

(3) שנה את התכנית MASKORET2 כדי שתמלא את הדרישות הבאות:
א. על שכר העבודה (למעט מתנות) חל מס הכנסה של 5.3%.
ב. התוצאות צריכות להופיע במבנה הבא (התו ^ יסמן תו רווח):

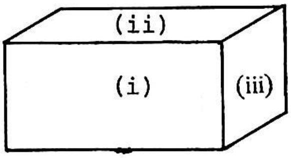
```
 שעות     שכר              סה"כ          מס          יתרה
עבודה^^^לשעה^^^מתנה^^^תשלומים^^^^^הכנסה^^^^לתשלום
-------------------------------------------------------
XXX XX,XXX XX,XXX X,XXX,XXX X,XXX,XXX X,XXX,XXX
```

(4) כתוב תכנית, אשר קוראת שני מספרים ומתייחסת אליהם כמידות של מלבן. הפלט יהיה ההיקף, השטח ואורך האלכסון של אותו מלבן. יש לכלול הערות בתכנית, כדי להסביר את המשתנים ואת השיטה של החישוב. הפלט צריך להיות משפט שיציג את משתני הקלט ואת משתני הפלט. המספרים יוצגו עם ספרה עשרונית אחת, כאשר הם מופרדים מן המלה שלידם ברווח אחד לפחות. הכן תרשים זרימה לפני שאתה מתחיל לתכנת.

(5) כתוב תכנית, אשר קוראת שני מספרים ומתיחסת אליהם כבסיס ואורך של היתר במשולש ישר זוית. הפלט יהיה גובה המשולש, הסינוס, הכוסינוס והטנגנס של הזוית שבין היתר לבין הבסיס. יש לכתוב הערות בכדי להסביר את המשתנים ואת שיטת החישוב. הפלט יהיה במבנה משפט שיציג את נתוני הקלט ואת הפלט. המספרים יוצגו עם ספרה עשרונית אחת, כאשר הם מופרדים מן המלה שלידם ע"י רווח אחד לפחות. הכן תרשים זרימה לפני כתיבת התכנית.

(6) כתוב תכנית שמקבלת כקלט את האורך, הרוחב והעומק בס"מ של תיבה מלבנית. כתוב תכנית שמחשבת את הערכים הבאים:

- השטח של הדפנות בצדדים.
- השטח של המכסה והבסיס.
- השטח של הדפנות מלפנים ומאחור.
- הנפח של התיבה.



אם מטר מרובע של נייר אלומיניום עולה 30 שקל, כמה יעלה לעטוף את כל הקופסה בנייר אלומיניום.

(7) תלמיד קיבל 3 ציונים במשך הסמסטר. כתוב תכנית בכדי לחשב את הממוצע של 3 הציונים, ההפרש של כל ציון מהממוצע, את הסטיה המוחלטת (ז"א הערך המוחלט של ההפרש) של כל ציון מן הממוצע ואת סטיית התקן של הציונים. סטיית התקן של 3 הציונים X1, X2, X3 מוגדרת כך:

$$\text{STIYAT_HATEKEN} = \sqrt{\frac{3(X1^2+X2^2+X3^2) - (X1+X2+X3)^2}{3*2}}$$

(8) כתוב תכנית שמקבלת כקלט את המספר N ומחשבת את השורש, הריבוע ואת המכפלה ב-5 וב-10, עבור 3 המספרים שלפני N ועבור 2 המספרים שאחרי N.

(9) נתון הרדיוס של עיגול. כתוב תכנית שמדפיסה את הקוטר, השטח וההיקף של אותו עיגול.

(10) סדרת פיבונצ'י מתחילה ב-0 וב-1. כל ערך נוסף בסדרה שווה לסכום של שני קודמיו. למשל, הערכים השלישי, הרביעי, החמישי, השישי, השביעי והשמיני בסדרה הם 0+1=1, 1+1=2, 1+2=3, 2+3=5, 3+5=8, 5+8=13. כתוב תכנית שמקבלת שני ערכי פיבונצ'י כלשהם עוקבים ומחשבת את שלושת הערכים הבאים בסדרה.

(11) כתוב תכנית שמקבלת כקלט ערך במעלות פרנהייט ומחשבת את מספר המעלות ביחידות צלסיוס. נוסחת המעבר היא: F = 9/5*C + 32.

(12) כתוב תכנית בכדי לחשב את השורשים של משוואה ריבועית AX**2+BX+C בעזרת הנוסחה $\frac{-B \pm\sqrt{(B^2-4AC)}}{2A}$ בהנחה שיש למשוואה שורשים ממשיים, ז"א ש: $B^2-4AC > 0$.

(13) כתוב תכנית בכדי לחשב את השטח של משולש בעל צלעות A, B, C (בהנחה שצלעות אלו באמת מהוות משולש) על סמך הנוסחה:

$$\text{שטח} = \sqrt{S(S-A)(S-B)(S-C)}, \quad \text{כאשר} \quad S=\frac{A+B+C}{2}$$

(14) כתוב תכנית שמקבלת את מספר החשבון בבנק, את היתרה הקודמת ואת כמות הכסף שבעל החשבון רוצה להפקיד או למשוך מחשבונו. כתוב תכנית בכדי להדפיס את היתרה החדשה בחשבון.

(15) כתוב תכנית בכדי לחשב את עלות הבנזין לנסיעה, כאשר נתונה העלות של ליטר בנזין, מרחק הנסיעה והקילומטרז' של המכונית לליטר בנזין.

(16) כתוב תכנית שמקבלת את משקלו של אדם בליברות ומדפיסה את משקלו באונקיות, בקילוגרמים ובגרמים. לידיעתך, ליברה אחת שווה ל-16 אונקיות וגם ל-0.45359 קילוגרמים.

(17) סוכן נכסי דלא ניידי מקבל דמי עמלה, כאחוז מסוים ממחיר הבית שהוא מוכר. המוכר מקבל את מחיר הבית פחות דמי העמלה ופחות הוצאות מכירה. כתוב תכנית שמקבלת כקלט את המחיר, את האחוז המגיע לסוכן ואת ההוצאות. התכנית מדפיסה את הסכום המגיע לידי המוכר.

כתוב תכניות עבור שני מקרים אחרים:

א. לא ידוע האחוז שהסוכן מקבל אבל שאר הפרטים ידועים. מצא את אחוז העמלה.

ב. לא ידועות ההוצאות, אבל שאר הפרטים ידועים. מצא את סכום ההוצאות.

(18) אדם משקיע P שקלים ומקבל I אחוז ריבית כל חודש. כמה כסף יהיה לו בסוף N חודשים. השתמש בנוסחה $S = P(1+I)^N$.

(19) חברת "מן המוכן" רוצה למחשב את החנות שלהם. המחירים בשקלים לקילוגרם הם:

- עוף - 3.50
- קציצה - 1.70
- כבד קצוץ - 6.80
- פשטידה - 0.60
- סלט ירקות - 0.84
- שניצל - 3.10

כתוב תכנית שתקבל כקלט את מספר הקילוגרמים מכל מוצר שהלקוח קונה. התכנית תדפיס את החשבון המפורט, שעליו יהיה כתוב בכמה עולה קילוגרם אחד של כל מצרך, כמה נקנה מאותו מצרך (בקילו) ובכמה עלתה הקנייה של אותו מצרך. בתחתית הרשימה יש לרשום את ערך כל הקניות ואח"כ את הסכום לתשלום כולל 15% מע"מ.

(20) משחק טוטו מתנהל בצורה זו: אנשים קונים טופס וממלאים אותו בתוצאות של משחקי כדורגל באותו סופשבוע, כפי שהם צופים אותם. הזוכים הם כל מי שניחשו נכונה את תוצאות המשחקים.
הרווחים ממכירת הטפסים מגיעים ל-3 יעדים:

- 10% נמסרים לממשלה.
- 15% הם דמי עמלה עבור הסוכן שמכר את הטופס.
- שאר הכסף מחולק באופן שווה בין האנשים שזכו.

כתוב תכנית בכדי לחשב כמה כסף יקבל כל זוכה, אם ידועה כמות הטפסים שנמכרו, מחיר הטופס ומספר הזוכים.

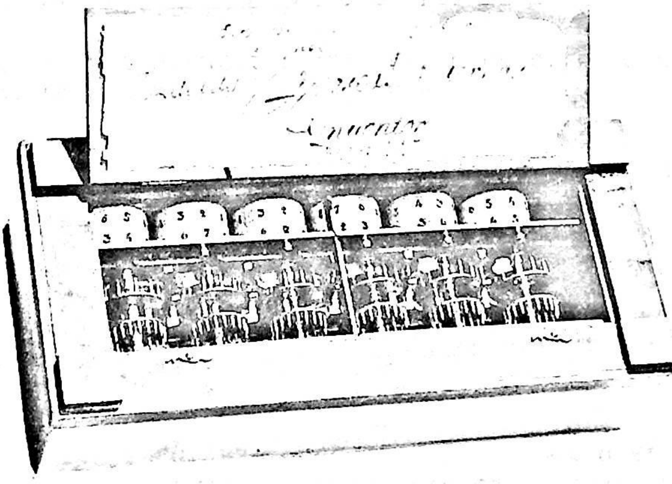
(21) נתונה מערכת של משוואות ליניאריות מן הצורה:

$$AX + BY = C$$
$$DX + EY = F$$

הפתרון הפשוט של מערכת זו הוא:

$$Y = \frac{AF - CD}{AE - BD} \quad , \quad X = \frac{CE - BF}{AE - BD}$$

כתוב תכנית שמקבלת כקלט את הערכים A,B,C,D,E,F ומדפיסה את הפתרון לזוג המשוואות. תוכל להניח שיש פתרון (מתי אין?).



מכונת החישוב של פסקל

פרק 2
משתנים וביטויים בוליאניים

2.1 משתנים בוליאניים

לעתים דרוש להעריך את היחס בין שני מספרים ולפעול בהתאם לתוצאה.

מה היא התשובה שהיית מצפה לקבל למשל, לשאלה זאת: "האם A שווה ל-B?" האפשרויות הן שהמספרים שווים, או שאינם שווים. אפשר גם להשיב: נכון (True) או לא-נכון (False) בהתאמה. בפסקל שומרים תשובה כזאת במשתנה בוליאני (Boolean Variable), אשר מקבל אחד משני הערכים שהזכרנו, TRUE או FALSE. במחשב, הערך "נכון" מיוצג ע"י הספרה 1 (אחד) והערך "לא-נכון" - ע"י הספרה 0 (אפס).

נעיין בתכנית הבאה:

```
PROGRAM HASHVAAH (INPUT, OUTPUT);
  (* קרא שני מספרים שלמים וקבע איזה מהם גדול יותר *)
VAR
  MISPAR1, MISPAR2: INTEGER;
  KATAN, SHAVEH, GADOL: BOOLEAN;
BEGIN
  WRITELN ('מהם המספרים שאתה רוצה להשוות?');
  WRITELN (' להקיש את הראשון ואת השני');
  READ (MISPAR1, MISPAR2);
  KATAN := MISPAR1 < MISPAR2;
  SHAVEH := MISPAR1 = MISPAR2;
  GADOL := MISPAR1 > MISPAR2 ;
  WRITELN ('HISHVENU^', MISPAR1, 'V^^':5, MISPAR2);
  WRITELN ('MISPAR1 < MISPAR2 IS^^', KATAN);
  WRITELN ('MISPAR1 = MISPAR2 IS^^', SHAVEH);
  WRITELN ('MISPAR1 > MISPAR2 IS^^', GADOL)
END.
```

הרצה של התוכנית הזו תפיק את הפלט הבא:

```
מהם המספרים שאתה רוצה להשוות ?
להקיש את הראשון ואת השני
 5        10
HISHVENU^5^^V^^10
MISPAR1 < MISPAR2 IS^^^TRUE
MISPAR1 = MISPAR2 IS^^FALSE
MISPAR1 > MISPAR2 IS^^FALSE
```

<u>הסבר:</u>

(1) שם התכנית.

(2) הצגה של מטרת התכנית.

(4) המשתנים MISPAR1 ו-MISPAR2 מקבלים את המספרים שנקרא כקלט ונשווה אותם זה לזה.

(5) כמו שמצהירים על משתנה שלם וממשי, כך יש להצהיר במשפט VAR גם על משתנה בוליאני. הכינוי BOOLEAN מחליף את הכינוי INTEGER או REAL, כדי לתאר את המשתנים. השמות של משתנים בוליאניים נקבעים לפי הכללים המקובלים בקביעת שמות בשפת פסקל.

המשתנים הבוליאניים האלה יקבלו במשך התכנית את הערך TRUE או את הערך FALSE. למשל, KATAN יקבל את הערך TRUE רק אם "<u>נכון</u>" ש-MISPAR1 קטן מ-MISPAR2, אחרת ערכו יהיה FALSE, שמשמעותו "<u>לא נכון</u>". המשתנה הבוליאני SHAVEH יקבל ערך TRUE רק אם MISPAR1 יהיה שווה ל-MISPAR2 והמשתנה הבוליאני GADOL יקבל את הערך TRUE רק אם MISPAR1 גדול מ-MISPAR2.

(9)-(7) קריאת המספרים להשוואה.

(12)-(10) בפסקל הסטנדרטי, משתנה בוליאני מקבל את ערכו רק ע"י פקודת השמה ולא ע"י קריאה מבחוץ, כקלט. בניבים שונים של השפה איפשרו לתת ערך למשתנה בוליאני גם ע"י פעולת קלט. בצד ימין של פקודת ההשמה נרשום:

- תנאי או תנאים שהמחשב יבדוק ויקבל עבורם תוצאת TRUE או FALSE.
- המלה TRUE או FALSE. הערך יאוחסן בכתובת של המשתנה הבוליאני.

(13) הכנת הפלט. כתבנו 5:'^^V' בכדי שיהיו שני רווחים בכל צד של התו V. בפקודה זו הוספנו שני רווחים בצד ימין של V. המספר 5 אחרי הנקודתיים יגרום לכך שישמרו 5 תווים עבור המחרוזת כולה, אך מכיון שיש רק 3 תווים בין הגרשיים, יתוספו שני רווחים בצד שמאל.

(14) הערך של המשתנה KATAN הוא TRUE או FALSE ולכן הפקודה הזאת תגרום לכתיבת אחד משני המשפטים האלה:

א. MISPAR1 < MISPAR2 IS ^TRUE
ב. MISPAR 1 < MISPAR2 IS FALSE

המהדר מקצה 5 תווים למלה TRUE (שתוצג עם רוח אחד משמאל) ולמלה FALSE. אפשר לשנות ע"י הוראות מפורשות את מספר התווים המוקצים למלים. למשל,

KATAN:10 יגרום להוספת 5 רווחים בצד שמאל של המלה FALSE, שאורכה 5, או להוספה של 6 רווחים משמאל למלה TRUE שאורכה 4.

(16)-(15) ביצוע דומה לזה שבשורה (14).

2.2 ביטויים בוליאניים

המחשב יכול לחשב תנאי פשוט (Simple Condition) הבנוי כך:

ביטוי אריתמטי   יחס   ביטוי אריתמטי

היחסים האפשריים הם:

=	שוויון
<>	אי-שוויון
>	גדול מ-
>=	גדול או שווה ל-
<	קטן מ-
<=	קטן או שווה ל-

עבור כל תנאי אפשר לומר, שהוא קיים (TRUE) או שהוא לא קיים (FALSE) ולכן הוא יכול להיות קלט למשתנה בוליאני. התנאי צריך להיות כזה, שאפשר יהיה לקבוע את נכונותו. כלומר, הערכים שמשווים צריכים להיות מאותו סוג. כפי שאי אפשר להשוות תפוחים עם תפוזים, כך אי אפשר להשוות משתנה בוליאני שערכו TRUE עם משתנה ממשי שערכו 7.52. מאידך, אפשר להשוות שני משתנים בוליאניים כי הערך FALSE נחשב כ-0 ולכן הוא קטן מן הערך TRUE שנחשב 1.

נניח ש-MISPAR1 שווה ל-5 ו-MISPAR2 שווה ל-10. הבדיקה MISPAR1<MISPAR2 תתן לנו את התוצאה TRUE שתאוחסן במשתנה KATAN. במקרה זה נקבל את הערך FALSE עבור שני המשתנים האחרים, SHAVEH ו-GADOL, בפקודות (11) ו-(12), בהתאמה.

ביטויים אריתמטיים אשר מהווים חלק מתנאי יכולים להכיל קבוע או משתנה אחד, או להיות יותר מורכבים, כמו למשל:

LACHATZ * (CHOM - 4 * PI) <> NEFACH - LACHATZ

אפשר לשלב יחד שני משתנים, קבועים או ביטויים אריתמטיים ע"י אופרטורים אריתמטיים (+,-,*,/,DIV,MOD) כדי לקבל ביטויים אריתמטיים מורכבים. כך גם אפשר לשלב יחד תנאים פשוטים בעזרת האופרטורים הבוליאניים AND, OR, NOT, כדי לקבל תנאים מורכבים (Complex Conditions).

הערך המיוחס לשני תנאים פשוטים המשולבים ע"י התנאי AND יהיה TRUE רק אם כל אחד משני התנאים יהיה TRUE. מאידך, הערך המיוחס לשני תנאים המשולבים ע"י OR יהיה FALSE רק אם כל אחד משני התנאים הוא FALSE. האופרטור NOT פועל על תנאי אחד בלבד והופך את הערך שלו מ-TRUE ל-FALSE ולהיפך.

נניח שהכנסנו את הערכים של שני תנאים שונים לתוך המשתנים A ו-B בהתאמה. הטבלה למטה, אשר נקראת טבלת אמת (Truth Table) מסכמת את השפעת האופרטורים השונים על המשתנים האלו.

A	B	A AND B	A OR B	NOT A
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

לדוגמא, נניח ש-A שווה ל-5, B שווה ל-10 ו-C שווה ל-2. אזי הביטוי

(A < 7*C) AND (B+7 < A+C)

מהווה תנאי מורכב. ערכו של התנאי הפשוט השמאלי הוא TRUE מפני ש-A, שערכו 5, באמת קטן מ-7*C, שערכו 14. התנאי הפשוט השני B+7<A+C הוא FALSE מפני שערכו של B+7 הוא 17, שאינו קטן מ-A+C שערכו 7. היות והתנאי השמאלי נכון והתנאי הימני לא-נכון ושניהם משולבים ע"י AND, הערך של התנאי המורכב הוא FALSE.

ביטוי אריתמטי הוא שילוב של אחד או יותר קבועים או משתנים מספריים ע"י אופרטורים אריתמטיים. בדרך דומה ניתן להגדיר ביטוי בוליאני (Boolean Expression) כשילוב של תנאי אחד או יותר ושל משתנים או קבועים בוליאניים (הכוונה למלים TRUE או FALSE), באמצעות אופרטורים בוליאניים. הערך של ביטוי אריתמטי יכול להיות כל ערך בתחום המספרים שהמחשב יכול להחזיק. לעומתו, הערך של ביטוי בוליאני מוגבל לשני ערכים בלבד, TRUE או FALSE.

על אף התחום המצומצם של הערכים, ביטוי בוליאני יכול להכיל יותר אופרטורים מאשר ביטוי אריתמטי, שהרי תנאי מכיל ביטויים אריתמטיים וגם אופרטורים של יחס. בביטוי בוליאני אפשר לשלב יחד כמה תנאים ע"י אופרטורים בוליאניים. בגלל המספר הרב של אופרטורים אשר אפשר לכלול בביטוי בוליאני, יש צורך להוסיף על חוקי הקדימות המקובלים, כפי שמפורט ברשימה שלהלן. ההרחבות הודגשו בקו:

(1) ביטוי בתוך סוגריים.
(2) - כאשר הוא סימן לשלילה, NOT .
(3) AND , * , / , DIV, MOD
(4) OR , + , -
(5) < , <= , = , <> , > , >=

כאשר מעריכים ביטוי מתקדמים משמאל לימין ומטפלים תחילה באופרטורים ברמה הגבוהה ביותר. כאשר יש להבדיל בין שני אופרטורים שנמצאים באותה רמה, הקדימות היא לפי סדר הרישום שלהם, משמאל לימין. אם נסיר את הסוגריים בדוגמא הקודמת נקבל:

A < 7*C AND B+7 < A+C

לפי חוקי הקדימות מעריכים תחילה את 7*C ומקבלים 14 (כי C=2). אח"כ מעריכים את: B AND 14, כאשר B=10. אפשר לראות שהביטוי הזה הוא לא חוקי כי האופרטור AND מתאים רק לשני אופרנדים בוליאניים ופה יש שני אופרנדים שהם ערכים שלמים. במצב הקודם הסוגריים חייבו הערכה קודמת של התנאים הפשוטים A<7*C ו- B+7<A+C והתוצאות הבוליאניות שולבו כחוק ע"י האופרטור AND.

השימוש בסוגריים אינו דרוש תמיד. אם נכתוב למשל, ביטוי כמו B<A+7 ונוסיף סוגריים בצורה הבאה: B<(A+7), לא תרמנו דבר, שהרי בכל מקרה מעריכים את A+7 תחילה. העובדה שמצב כזה רצוי ומונע שימוש יתר של סוגריים, הניעה את מחברי השפה לקבוע את קדימותם של היחסים כנמוכה ביותר.

נכתוב תכנית אשר יוצרת את טבלת האמת שראינו קודם.

```
(1)   PROGRAM TAVLAT_EMET (OUTPUT);
(2)   (* בתכנית הזאת בונים טבלת אמת. יש רק פלט *)
(3)   VAR
(4)      P, Q : BOOLEAN;
(5)   BEGIN
(6)      WRITELN ( '^^P^^^^^^^^', '^^Q^^^^^^^^',
(7)         'P^OR^Q^^^^', 'P^AND^Q^^^',
(8)         'NOT^P^^^^^');
(9)      WRITELN;
(10)     P:= TRUE;   Q:= TRUE;
(11)     WRITELN (P, '^':5, Q, '^':5, P OR Q, '^':5,
(12)        P AND Q, '^':5, NOT P, '^':5);
(13)     P:= TRUE;   Q:= FALSE;
(14)     WRITELN ( P, '^':5, Q, '^':5, P OR Q, '^': 5,
(15)        P AND Q, '^':5, NOT P, '^':5);
(16)     P:= FALSE;  Q:= TRUE;
(17)     WRITELN (P, '^':5, Q, '^':5, P OR Q, '^':5,
(18)        P AND Q, '^':5, NOT P, '^':5);
(19)     P:= FALSE;  Q:= FALSE;
(20)     WRITELN (P, '^':5, P OR Q, '^':5,
(21)        P AND Q, '^':5, NOT P, '^':5)
(22)  END.
```

הפלט של התכנית נראה כך:

```
^^P^^^^^^^^^^Q^^^^^^^^^P OR Q^^^^P AND Q^^^NOT P^^^^^

^TRUE      ^TRUE      ^TRUE      ^TRUE      FALSE
^TRUE      FALSE      ^TRUE      FALSE      FALSE
FALSE      ^TRUE      ^TRUE      FALSE      ^TRUE
FALSE      FALSE      FALSE      FALSE      ^TRUE
```

<u>הסבר</u>:

בתכנית זו כתבנו במספר שורות (10, 13 ועוד) שתי פקודות בכל שורה. אפשר לכתוב שתי שורות בשורה אחת ואף יותר, אך צריך להפריד ביניהן בסימן נקודה-פסיק. בכל זאת מומלץ לכתוב בכל שורה פקודה אחת בלבד.

(1)-(2) שם התכנית ותיאורה.

(3)-(4) הצהרה על משתנים.

(6)-(8) הדפסת הכותרת של הטבלה. שים לב שעל אף שכתוב פה P OR Q, P AND Q ו-NOT P פעולות אלו לא מתבצעות, אלא מודפסות כמלל, מכיוון שהן רשומות בין גרשים.

(9) WRITELN גורם לדילוג שורה בין הכותרת ותוכן הטבלה.

(10)-(12), (13)-(15), (16)-(18), (19)-(21) חוזרים על אותן פקודות לכל אחד מ-4 התמורות של TRUE ו-FALSE. למשל, בתמורה הראשונה גם P וגם Q שווים ל-TRUE. לכן מדפיסים P ו-5 רווחים, Q ו-5 רווחים, <u>הערך</u> של

P OR Q (שהוא TRUE, במקרה זה) ו-5 רווחים, <u>הערך</u> של P AND Q (TRUE במקרה זה) ו-5 רווחים והערך של NOT P (FALSE במקרה זה) ו-5 רווחים.

להדפסת הכותרת של הטבלה (שורות 6-8) מקיפים את שמות המשתנים, או את תיאור הפעולות הבוליאניות בגרשים. כאן אין עושים זאת, כי לא מדפיסים את הכינויים אלא את התוכן של המשתנים, או את תוצאות הפעולות שביניהם.

2.3 חוקי די-מורגן (*)

חוקי די-מורגן מגדירים יחסים שונים בין משתנים בוליאניים. בכך הם יכולים לסייע בניסוח הפעולות הבוליאניות. חוקים אלה קובעים שאם A ו-B הם משתנים בוליאניים, יש זהות בין היחסים הבאים:

(1) NOT (A AND B) ו- (NOT A) OR (NOT B)
(2) NOT (A OR B) ו- (NOT A) AND (NOT B)

בנוסף, כדאי לזכור שיש זהות בין הביטויים הבאים:

(3) NOT (NOT A) ו- A
(4) NOT (A = B) ו- (A <> B)
(5) NOT (A < B) ו- (A >= B)
(6) NOT (A > B) ו- (A <= B)

לדוגמא, נפשט את הביטוי הבוליאני הבא:

NOT ((A < B) AND (C <> D))

בעזרת זהות (1) נכתוב את הביטוי כך:

NOT (A < B) OR NOT (C <> D)

בעזרת זהויות (4) ו-(5) נוכל לכתוב את הביטוי החדש כך:

(A >= B) OR NOT (NOT (C = D))

בעזרת (3) נקבל:

(A >= B) OR (C = D)

2.4 תרגילים

2.4.1 שאלות

(1) אם A=TRUE, B=FALSE, C=FALSE ו- D=FALSE, עליך להעריך את הביטויים הבאים. רשום גם את סדר ההערכה.

(a) A OR B AND C OR D
(b) A AND B OR C AND D
(c) A AND NOT B OR NOT A AND B
(d) A = B OR C >= D
(e) (A = B) OR (C >= D)
(f) A <> B AND C <> D
(g) (A <> B) AND (C <> D)

(2) * פשט את הביטוי הבוליאני הבא:
NOT ((NOT A OR (X < Y)) AND (P = Q)
כאשר A הוא משתנה בוליאני ו-Q, P, Y, X הם משתנים שלמים.

(3) כתוב ביטוי בוליאני שערכו TRUE אם:
- I מתחלק ב-J ללא שארית.
- I אי זוגי.
- I איננו בין 5- ל: 4- וגם איננו בין 4+ ל: 5+.

(4) אם I=4, J=8 איזה מבין הביטויים ערכם TRUE?
- 2 * I < J
- 2 * I-1 < J
- (I > 0) AND (I <= 10)
- (I > 25) OR (I < 50) AND (J < 50)
- (I < 4) OR (J > 5)
- NOT (I > 6)

2.4.2 תכניות

(1) רצוי שעובד חדש בחברה ימלא את 3 הדרישות דלהלן:
- למעלה מגיל 21.
- בעל תואר ראשון.
- בעל ניסיון של 5 שנים או יותר.

כתוב תכנית שתבדוק אילו מבין דרישות אלו המועמד ממלא. הקלט יהיה 3 מספרים שלמים:
- גיל המועמד.
- תואר: 0 יסמל שאין לו תואר.
  1 יסמל שיש לו תואר.
- מספר שנות ניסיון.

כפלט נפיק בעבור כל מועמד 4 שורות במבנה הבא:

```
HAMUAMAD MEMALEH KOL HADRISHOT: *****
HAMUAMAD MEMALEH ALEPH VBET OH GIMEL: *****
HAMUAMAD MEMALEH BET OH GIMEL: *****
HMUAMAD LO MEMALEH AF AHAD MIN HADRISHOT: *****
```

***** - הכוכביות מסמנות את המלה TRUE או FALSE לפי המקרה.

(2) כתוב תכנית שתקרא שני זוגות של מספרים שלמים ותדפיס FALSE או TRUE עבור כל אחד מן התנאים הבאים:
א. האם שני הזוגות זהים?
ב. האם האיבר הראשון או האיבר השני שווה בשני הזוגות?
ג. האם האיבר הראשון בזוג הראשון שווה לאיבר השני בזוג השני, או האם האיבר השני בזוג הראשון שווה לאיבר הראשון בזוג השני?

(3) חברת אל-על רוצה לגייס אנשי ביטחון מקרב תלמידי האוניברסיטה. יש לה דרישות דלהלן:

- זכר.
- גמר 85 נקודות לקראת תואר.
- גובה למעלה מ-1.8 מטרים.
- משקל מעל ל-70 קילו.

הקלט מורכב מ-5 נתונים :

- מספר סטודנט.
- מין (1 - זכר, 0 - נקבה)
- מספר נקודות שהוא גמר.
- גובה.
- משקל.

כתוב תכנית שתדפיס את שורת הפלט הבאה עבור כל סטודנט:

סטודנט מספר NNNN מתאים להיות איש ביטחון: XXXXX.

התווים NNNN יוחלפו ע"י מספר הסטודנט והתווים XXXX יוחלפו ע"י המלים TRUE או FALSE לפי המקרה.

(4) חברה להפצת ספרים מקבלת הזמנות ושולחת ספרים. לקוח קבוע משלם רק כאשר הוא מקבל את הספרים בדואר ("טיפול מיוחד"). לקוח רגיל חייב לשלם לפני ששולחים לו את הספרים ("טיפול רגיל"). לקוח נחשב לקבוע אם הוא מקיים את התנאים דלהלן:

א. קונה סחורה שערכה עולה על 8,000 שקל לשנה.
ב. בעבר שילם את כל החשבונות בזמן (תוך 30 יום).
ג. בעל ותק של יותר מ-5 שנים.

חברת ההפצה מוכנה לתת טיפול מיוחד גם ללקוחות שמקיימים את התנאים דלהלן:

(1) (א) ו-(ב).
(2) (א) ו-(ג).
(3) (ב) ו-(ג).
(4) (ב).

כתוב תכנית שתקבל כקלט:

א. ערך הסחורה שהלקוח קונה במשך שנה.
ב. האם הלקוח שילם את החשבונות בזמן בעבר (1 - כן, 0 - לא).
ג. מספר השנים שהלקוח קונה ספרים מחברת הספרים.

הודעת הפלט צריכה להיות:

תן ללקוח טיפול מיוחד: XXXXX.

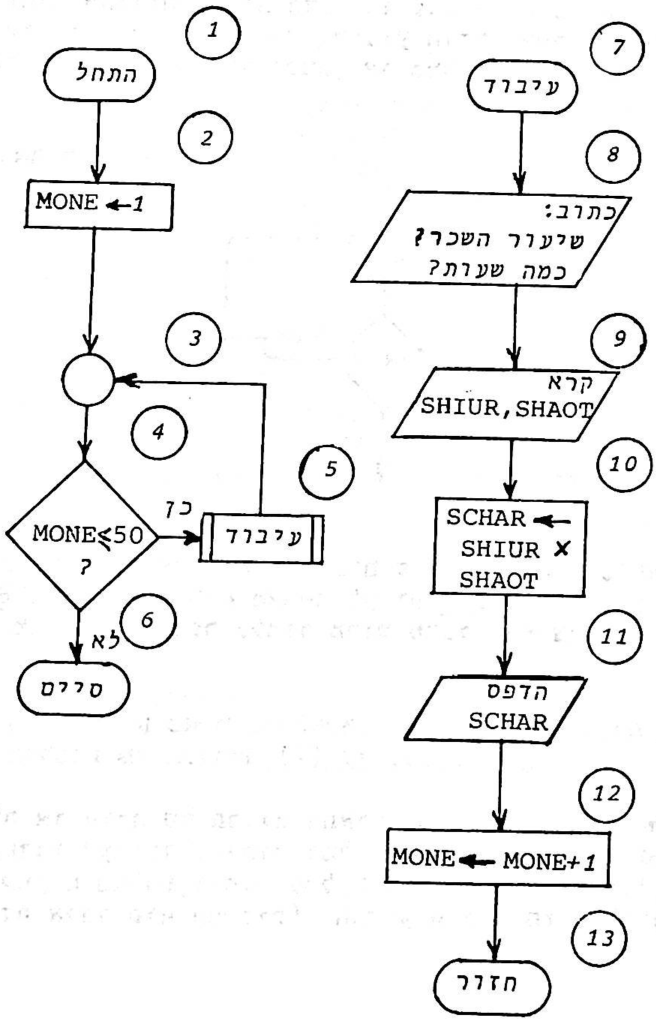
התווים XXXXX יוחלפו ע"י המלים TRUE או FALSE, לפי המקרה.
הערה: לפני כתיבת התכנית צמצם את מספר התנאים ל-2.

פרק 3

לולאות

3.1 הפקודה WHILE...DO

נניח שרוצים לחשב את המשכורת של כל הפועלים שבמפעל ולא של פועל אחד בלבד. נכין תרשים זרימה לפני שנתחיל לתכנת.



הסבר:

בתרשים זה יש סמלים, או בלוקים שונים, אשר נסביר את התפקיד של כל אחד מהם. המספרים בראשית הפיסקאות מתיחסים לסמלים שבתרשים.

(2) הגדרנו משתנה בשם MONE שערכו ההתחלתי הוא 1. בהמשך, בכל פעם שנקרא את הנתונים של פועל ונעבד אותם, נגדיל את MONE ב-1. לאחר שנעבד את הנתונים של פועל מספר 50, הערך של MONE יגיע ל-51 וזה יסמן לנו להפסיק את העיבוד ולסיים את התכנית.

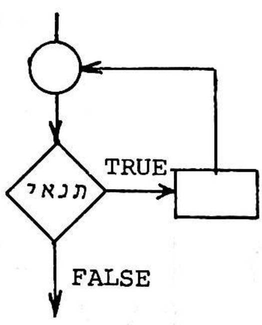
(3) מחבר, או נקודת קשר. אסור שחץ אחד יפגש עם חץ שני אלא בנקודת קשר. אסור ליותר משני חצים להפגש באותה נקודת קשר.

(4) מעוין מסמל החלטה. בנקודה זו צריך להחליט אם להמשיך הלאה לכיוון שמתאים למצב MONE>50, או להסתעף בכיוון ימין, שמתאים למצב MONE<=50.

(5) תהליך מוגדר מראש, או קטע. אם MONE<=50 מסתעפים למלבן זה, שכתוב בו "עיבוד". יש לו צלעות צד כפולות, אשר מסמלות שתהליך העיבוד הרשום בו מורכב ממספר פעולות או שלבים, המפורטים בתרשים זרימה נפרד.

(3), (4) ו-(5) יחד מהווים מבנה חדש. עד עכשיו רשמנו את הסמלים זה לאחר זה במבנה סדרתי. המבנה החדש מאפשר ביצוע חוזר, אשר תמיד מורכב מסמל החלטה, סמל של תהליך חישוב או השמה, או סמל של תהליך מוגדר מראש (קטע) ומחבר.

המבנה החדש נראה כך:



המבנה המוצג בתרשים הוא אחד משני מבנים של איטרציה (Iteration - מלשון חזרה) ונקרא DOWHILE, כי הוא מאפשר לעשות עיבוד (Do) רק כאשר (While) התנאי מתקיים. מן הציור אפשר ללמוד מדוע מבני איטרציה נקראים לולאות (Loops).

(6) מגיעים לסוף התכנית רק כאשר MONE=51, ז"א לאחר שמבצעים את הלולאה 50 פעמים. הלולאה מורכבת מן השלבים (7) עד (13).

(7) הסמל של התחלה או סיום של תכנית מתאים גם לסימול קטע, אשר מהווה חלק תוכנית נפרד שקרוי "עיבוד". בתוך סמל ההתחלה כותבים את שם הקטע (שהרי לפעמים יש כמה קטעים כאלה) ובתוך סמל הסיום כותבים "חזור" במקום לכתוב "סיום" (שהרי זה איננו סוף התכנית). המשמעות היא חזרה לתוכנית הראשית,

לנקודה שממנה הסתעפנו ל"עיבוד" ומשם להמשיך את הביצוע. בתוכנית זו החזרה היא למחבר אשר מסומן במספר 3.

(12) מטרת פעולה זו לעדכן את המונה, ז"א להגדיל אותו ב-1, כי ביצענו את הקטע פעם נוספת. כמו שהוזכר קודם, כאשר נגדיל את המונה מ-50 ל-51 ונחזור במהלך הביצוע ל-(3) ואח"כ ל-(4), תוצאת ההחלטה ב-(4) תהיה שונה ממה שהיתה עד עכשיו. במקום להמשיך ל-(5), נמשיך ל-(6), מכיון שעכשיו MONE>50.

תכנית המבוססת על תרשים זרימה זה מוצגת להלן:

```
 (1)  PROGRAM MASKORET3 (INPUT, OUTPUT);
 (2)       (* בתכנית הזאת מחשבים את המשכורת
 (3)                 של כל 50 הפועלים במפעל *)
 (4)  CONST
 (5)     MISPAR_OVDIM = 50;
 (6)  VAR
 (7)     SHIUR, SHAOT, SCHAR: REAL;
 (8)     MONE : INTEGER;
 (9)  BEGIN
(10)     MONE := 1;
(11)     WHILE MONE <= MISPAR_OVDIM   DO
(12)        BEGIN
(13)            WRITELN  ('מהו שיעור השכר לשעה? כמה שעות עבד?');
(14)            READLN  (SHIUR, SHAOT);
(15)            SCHAR := SHIUR * SHAOT;
(16)            WRITELN  (SCHAR:8:2, '^^מגיע לו שכר',
(17)                      'עבור^^',SHAOT:6:1,'^שעות עבודה');
(18)            MONE := MONE + 1
(19)        END
(20)  END.
```

קלט לדוגמא עבור אחד מהמחזורים נראה כך:

```
מהו שיעור השכר לשעה? כמה שעות עבד?
3.5    42
עבור^^42.0^^שעות עבודה מגיע לו שכר^^147.00^^
```

<u>הסבר:</u>

(1)-(3) שם התכנית.

(4)-(5) במקום לרשום את המספר 50 בחלק הביצועי של התוכנית, נגדיר תחילה משתנה MISPAR_OVDIM ונשתמש בו בהמשך. נוהג זה יתרום להבנת התוכנית ויאפשר לבצע אותה בעבור מספר כלשהו של עובדים לאחר שינוי הערך של הקבוע הזה.

(6)-(8) הצהרת המשתנים. המשתנים הממשיים SHIUR, SHAOT ו-SCHAR מוגדרים כמו בדוגמאות הקודמות. MONE הוא משתנה שלם שיוגדל ב-1 בכל פעם שנסיים מחזור אחד של ביצוע של הלולאה.

(10) השמה של הערך ההתחלתי של MONE, אשר יהיה שווה ל-1 עד שנגמור את הביצוע של הלולאה בפעם הראשונה. אח"כ נשנה אותו ל-2, 3 וכו', עד 51 (שים לב, 51 ולא 50, כי הוספנו 1 למונה לאחר שביצענו את הקטע בפעם ה-50).

(11)-(19) זו פקודה המקבילה למבנה DOWHILE. צורתה הכללית היא:



כאשר ערכו של הביטוי שווה ל-TRUE, מבצעים את הקטע. כאשר ערכו שווה ל-FALSE לא מבצעים את הקטע, אלא ממשיכים בפקודה הבאה.

הביטוי הבוליאני בתוכנית זו הוא: MONE <= MISPAR_OVDIM. בפקודה (10) נקבע למשתנה MONE ערך התחלתי של "1". בביצוע הראשון, אם-כן, התנאי MONE <= MISPAR_OVDIM אכן אמת ולכן נבצע את הקטע. אבל איפה הקטע? כאשר ציירנו תרשים זרימה, הקטע נרשם בנפרד. בקידוד קיימת אפשרות כזאת (שנלמד עליה בפרק 8), אבל גם קיימת אפשרות לכתוב את הקטע במקומו, כאשר מציינים את תחילתו וסופו ע"י המלים BEGIN ו-END, בהתאמה.

קטע אשר מופיע בגוף התכנית נקרא <u>פקודה מרוכבת</u> (Compound Statement), אשר בתכנית זו הוא נמשך מפקודה (12) עד פקודה (19). אחת מן הפקודות הכלולות בקטע צריכה להשפיע על הביטוי הבוליאני, כדי שערכו יהפוך מ-TRUE ל-FALSE בשלב כלשהו של העיבוד. כך, בסופו של דבר, נפסיק לבצע את הלולאה. אחרת נקבל לולאה אין-סופית. בדוגמא זו הפקודה שעושה זאת היא פקודה (18), אשר מוסיפה "1" למונה כל פעם שעוברים דרך הקטע.

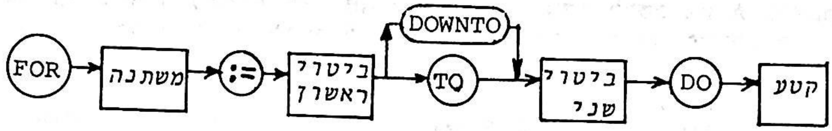
לאחר שמבצעים את הקטע 50 פעמים, MONE יהיה שווה ל-51, הביטוי הבוליאני יפסיק להיות TRUE, וביצוע התכנית ימשיך בפקודה הבאה שלאחר סיום הקטע.

3.2 הפקודה FOR...DO

בתכנית הקודמת השתמשנו בשלוש פקודות שונות בכדי ליצור את הלולאה ולפקח על פעולתה. לכל פקודה יש תפקיד נפרד:

- בפקודה (10) קבענו ערך התחלתי למונה,
- בפקודה (18) קידמנו את המונה כדי שבסופו של דבר הביטוי הבוליאני יהפך מ-TRUE ל-FALSE,
- בפקודה (11) הערכנו את הביטוי הבוליאני.

קיימת פקודה אחת שמבצעת, מבחינתו של המתכנת, את שלוש הפעולות האלה וחוסכת ממנו את הטירחה הכרוכה בכתיבה של פקודות נפרדות. זוהי הפקודה FOR, שצורתה הכללית היא:



המהדר בונה בעצמו את הפקודות לבקרת הלולאה. פקודות אלו לא נראות למתכנת, אשר ממשיך לראות רק את הפקודות שכתב.

המשתנה מקבל כערך התחלתי את הערך של הביטוי הראשון ואז מבוצע הקטע לראשונה. בסיום הביצוע של הקטע גדל הערך של המשתנה ב-1 אם כתבנו TO, או שהוא קטן ב-1 אם כתבנו DOWNTO. בשלב זה נערכת בדיקה אם הערך החדש של המשתנה עוד לא גדול מן הביטוי השני (אם כתבנו TO), או אם ערכו עוד לא קטן מן הערך של המשתנה השני (אם כתבנו DOWNTO). אם התשובה היא <u>כן</u>, ממשיכים לבצע את הלולאה, אחרת ממשיכים עם הפקודה הבאה בתכנית.

אם נשתמש בפקודה החדשה FOR, נוכל להשמיט בתוכנית הקודמת את הפקודות (10) ו-(18) ולהחליף את הפקודה (11) בפקודה:

(11a) FOR MONE := 1 TO MISPAR_OVDIM DO

אם רוצים להתחיל עם ערך גדול עבור MONE ולהקטין אותו נוכל לכתוב:

(11b) FOR MONE := MISPAR_OVDIM DOWNTO 1 DO

המשתנה MONE הוא <u>האינדקס של הלולאה</u> FOR והוא יכול להופיע גם בתוך הלולאה. למשל, אם נכניס לפני פקודה (16) את הפקודה WRITELN(MONE), נוכל להדפיס את המספר הסידורי של כל פועל ליד המשכורת שלו. מאידך, אסור לשנות את ערכו של האינדקס בתוך הלולאה, שהרי עדכון האינדקס נעשה באופן אוטומטי ע"י FOR ואיננו נמצא בשליטת המתכנת.

לדוגמא, אם נכתוב MONE:=MONE+1 בתוך לולאת FOR, אזי MONE יתעדכן פעמיים. פעם אחת יבוצע העדכון כתוצאה מהפקודה הנזכרת ופעם אחרת - באופן אוטומטי בסוף הלולאה. לכן מומלץ להמנע מעדכון של אינדקס הלולאה בתוך הלולאה.

אם נוסיף אחרי פקודה (9) את הפקודה READLN (MISPAR_OVDIM) ונצהיר על MISPAR_OVDIM כמשתנה שלם ולא כקבוע, נוכל לשנות את הפקודה (11) כך:

(11c) FOR MONE := 1 TO MISPAR_OVDIM

בתוכנית זו אין צורך לדעת את מספר הפועלים בזמן כתיבת התכנית, אלא רק בזמן הביצוע.

<u>3.2.1 חישובי חזקות וטור טיילור</u> (*)

הפקודה FOR, כפי שראינו, כוללת בתוכה את המנגנון לביצוע תפקידים של שלוש הפקודות לבקרת הלולאה. יתכן מצב שבו הקטע שחוזרים עליו יהיה מורכב מפקודה אחת בלבד ואז אין צורך במלות העזר BEGIN ו-END, כמו שנראה בדוגמא הבאה.

בדוגמא הבאה נציג דרך נוספת לזו שהוצגה לעיל, כדי לחשב את A בחזקת B עבור מספר שלם (שהרי אין אפשרות לחשב חזקות באופן ישיר בשפת פסקל, כמו שהזכרנו כבר). B הוא מספר שלם.

```
A_BECHZKAT_B := 1;
FOR I := 1 TO B DO
    A_BECHEZKAT_B := A_BECHZKAT_B * A;
```

נוכל גם להשתמש בפקודה FOR..DO בכדי לחשב את ערכו של e^x, אשר מוגדר ע"י סדרת טיילור. שים לב, ככל ש-n יותר גדול, התוצאה יותר מדויקת.

$$e^x = 1 + \frac{X}{1!} + \frac{X^2}{2!} + \frac{X^3}{3!} +..+ \frac{X^n}{n!}$$

בפועל, אין צורך בתכנית לחישוב e^x, כי הפונקציה EXP(X) אשר מחשבת זאת מוגדרת כחלק משפת פסקל. התוכנית שלהלן מוצגת בכדי להדגים שיטה ולא מכיון שהיא דרושה.

```
 (1)  PROGRAM EXP (INPUT, OUTPUT);
 (2)  VAR
 (3)      MISPAR_EVARIM, I : INTEGER;
 (4)      X, SCHUM, EVAR : REAL;
 (5)  BEGIN
 (6)    WRITELN ('לדיוק של כמה איברים נחשב את ex ומה ערכו של X?');
 (7)    READLN (MISPAR_EVARIM, X);
 (8)    EVAR := 1;
 (9)    SCHUM := 1;
(10)    FOR I := 1 TO MISPAR_EVARIM DO
(11)      BEGIN
(12)        EVAR := EVAR * X / I;
(13)        SCHUM := SCHUM + EVAR
(14)      END;
(15)    WRITELN (MISPAR_EVARIM, '^^בדיוק של^^ ', X:6:2,'^^בחזקת E');
(16)    WRITELN (SCHUM:8:2, '^^:איברים, שווה ל')
(17)  END.
```

הרצה של התוכנית תוכל להפיק את הפלט הבא:

```
לדיוק של כמה אברים נחשב את ex ומה ערכו של X?
7, 4.00
7^^בדיוק של^^^^4.00^^בחזקת E
^^^51.80^^:איברים, שווה ל
```

<u>הסבר:</u>

(1) שם התכנית.

(2)-(4) המשתנה X הוא החזקה. המשתנה EVAR הוא הערך של כל איבר חדש שמוסיפים לסכום המצטבר, SCHUM. הדיוק של התוצאה נובע ממספר האיברים שישתתפו בסכום, MISPAR_EVARIM. I משמש כאינדקס עבור הפקודה FOR..DO.

(6)-(7) קוראים ערכים למשתנים MISPAR_EVARIM ו-X, אשר יקבעו בהתאמה את מידת הדיוק ואת החזקה X שנעלה בה את הבסיס e.

(9)-(8) ערכים התחלתיים של EVAR ו-SCHUM. הערך ההתחלתי של SCHUM הוא 1 (אחד) וזהו האיבר הראשון של סדרת טיילור.

(14)-(10) מבצעים לולאה. בכל פעם שמבצעים את הלולאה, מוסיפים איבר אחד מן הסדרה עד שנוסיף את הערך של האיבר האחרון (ז"א האיבר שמספרו MISPAR_EVARIM) לסכום ההתחלתי שערכו 1. בפעם הראשונה שמבצעים את הלולאה מחשבים את X/1 ומוסיפים אותו ל-SCHUM. בפעם השנייה מחשבים (X/1)*(X/2) אשר שווה ל-$X^2/2!$ ומוסיפים אותו ל-SCHUM וכו'. האיבר האחרון שווה לביטוי: (X/1)*(X/2)*..*(X/MISPAR_EVARIM) אשר שווה לביטוי: $X^{mispar_evarim}/$ MISPAR_EVARIM! ומוסיפים אותו ל-SCHUM.

(16)-(15) הדפסת התוצאה, יחד עם החזקה X ומידת הדיוק MISPAR_EVARIM. יש לשים לב לכך שההודעה היא בעברית, לכן כתבנו WRITELN בשורה (15) וגם בשורה (16), כדי לשמור על הסדר הנכון של המשפט. המחרוזת הראשונה מסתיימת בסוף שורה (15). אם לא היינו כותבים פקודת WRITELN בשורה (16), שתי השורות היו מצורפות זו לזו והשורה (15) היתה מופיעה בצד שמאל של השורה (16) בסדר לא נכון. הפקודה WRITELN בשורה (16) גורמת להדפסה נפרדת של התמליל שבשתי שורות אלו. בכתיבה באנגלית בעיה זו אינה קיימת, כי התמליל שבשורה השנייה הינו ברצף של השורה הראשונה, משמאל לימין.

3.3 לולאה ללא גבול עליון

על אף היעילות והנוחות של השימוש בפקודה FOR, לא תמיד היא עדיפה על הפקודה WHILE..DO. בדוגמא של חישוב A בחזקת B, מספר הלולאות היה ידוע בזמן כתיבת התכנית. גם בדוגמת המשכורת, כאשר השתמשנו בפקודה (11c), היינו צריכים לדעת בזמן ביצוע התוכנית כמה לולאות (ערכו של MISPAR_OVDIM) רוצים לבצע. מאידך, אם מספר זה לא ידוע בזמן ביצוע התכנית, אי אפשר להשתמש בפקודת FOR.

דוגמא של מצב כזה, ז"א כאשר לא ידוע מספר הלולאות בתחילת הביצוע, נראה בתוכנית הבאה:

```
PROGRAM MASKORET4 (INPUT, OUTPUT);
      (* התכנית הזאת מחשבת משכורת פועלים.
         לא צריכים לדעת מראש כמה פועלים יש. *)
CONST
   SOF_NETUNIM = -1.0;
            (* כאשר הערך של SHIUR יהיה -1.0, הוא יציין
               שהגענו לסוף קלט הנתונים. *)
VAR
   SHIUR, SHAOT, SCHAR: REAL;
BEGIN
     WRITELN ('מהו שיעור השכר לשעה?');
     READLN(SHIUR);
     WHILE SHIUR <> SOF_NETUNIM DO
        BEGIN
```

```
(15)            WRITELN ('כמה שעות עבד?');
(16)            READLN (SHAOT);
(17)            SCHAR := SHIUR * SHAOT;
(18)            WRITELN  ('מגיע לו שכר של', SCHAR:8:2, 'שקל',
(19)                      'בעבור', SHAOT:6:1, 'שעות עבודה');
(20)            WRITELN ('מהו שיעור השכר לשעה?');
(21)            READLN (SHIUR)
(22)         END
(23)   END.
```

<u>הסבר</u>:

(1)-(3) שם התוכנית ותיאור.

(4)-(7) הגדרת הערך שישמש <u>כזקיף</u> (Flag, דגל) בכדי להפסיק את ביצוע הלולאה. ערך זה איננו יכול להיות אחד מהערכים ש-SHIUR יכול לקבל. כאשר נקרא אותו הוא לא יחשב כערך עבור המשתנה SHIUR, אלא כסימן לסיום הנתונים ולהפסקת ביצוע הלולאה. בתכנית זו נבחר הערך 1.0- למטרה זו, כי SHIUR חייב להיות גדול מ-0.

(8)-(9) הצהרה על המשתנים שישתתפו בתוכנית.

(11)-(12) קריאת הערך של SHIUR בפעם הראשונה. גם לפני התחלת ביצוע הלולאה חייבים ליחס ערך למשתנה SHIUR, כי הוא מופיע כחלק מן הביטוי הבוליאני שבתחילת הלולאה. אם SHIUR לא יהיה מוגדר כנדרש גם לפני הביצוע הראשון של הלולאה, יש חשש שלא נכנס אליה כלל.

(13)-(22) פקודות אלו מהוות את הלולאה. ערכו של SHIUR עבור הקריאה הראשונה ידוע כבר בתחילת ביצוע הלולאה. אם למשל לא יהיו נתונים, SHIUR יקבל את הערך 1.0- כבר בשורה (12) והלולאה לא תתבצע אפילו פעם אחת. אבל זהו מקרה יוצא דופן, שהרי אם אין נתונים, לא מריצים את התוכנית.

המקרה היותר סביר הוא ש-SHIUR יקבל ערך חיובי בשורה (12), נקרא את הערך של SHAOT בשורה (16), נחשב את השכר בשורה (17), נדפיסו בשורות (18)-(19) ולבסוף נקרא את ערכו החדש של SHIUR בשורות (20)-(21).

מקפידים לקרוא את SHIUR בסוף הלולאה כדי שאם ערכו יהיה שווה ל: SOF_NETUNIM נוכל לסיים מיד את ביצוע הלולאה, או ליתר דיוק, להמנע מהתחלה נוספת שלה. כאשר: SHIUR=SOF_NETUNIM, ערכו של התנאי הבוליאני בפקודה WHILE יהיה שווה ל-FALSE והלולאה תפסק.

בתוכנית הקודמת כתבנו פעמיים את הפקודה READ(SHIUR). בפעם הראשונה כתבנו אותה לפני התחלת הלולאה, כדי שיהיה למשתנה ערך מוגדר בתחילת הלולאה. בפעם השנייה כתבנו אותו בסוף הלולאה כדי לקבל ערכים לחישוב במחזור הבא ובכדי לסיים את הלולאה כאשר ערכו של SHIUR יהיה שווה ל-SOF_NETUNIM.

אם מניחים שיש בתכנית לפחות נתון אחד (וזה סביר), נוכל לפשט אותה, כפי שמוצג בתוכנית הבאה:

```
(1)  PROGRAM MASKORET5 (INPUT,OUTPUT);
```

```
(2)           התוכנית הזאת מחשבת משכורות של פועלים. *)
(3)                 (* לא צריכים לדעת את מספר הפועלים.
(4)    CONST
(5)           SOF_NETUNIM = 0;
(6)    VAR
(7)           SHIUR,SHAOT,SCHAR: REAL;
(8)           ZAKIF: INTEGER;
(9)    BEGIN
(10)       ZAKIF:= 1;
(11)       WHILE  ZAKIF <> SOF_NETUNIM  DO
(12)         BEGIN
(13)            WRITELN ('מהו שיעור השכר לשעה? כמה שעות עבד');
(14)            READLN (SHIUR, SHAOT);
(15)            SCHAR:= SHIUR * SHAOT;
(16)            WRITELN  (SCHAR:8:2, 'שכרו הוא^',
(17)                           'שעות עבודה^', SHAOT:6:1, 'עבור^^');
(18)            WRITELN ('האם יש עוד נתונים?');
(19)            WRITELN ('אם יש כתוב 1, אם אין כתוב 0');
(20)            READLN (ZAKIF)
(21)         END
(22)   END.
```

הרצה של תכנית זו זהה להרצה של התוכנית MASKORET4, אולם בסוף כל לולאה תוצגנה על המסך השאלה וההנחיה:

האם יש עוד נתונים?
אם יש כתוב 1, אם אין כתוב 0

<u>הסבר:</u>

(1)-(3) שם התוכנית ותיאור.

(4)-(5) המשתנה SOF_NETUNIM שונה פה מן הדוגמא הקודמת. בתכנית MASKORET4 הוא מייצג ערך ש-SHIUR מקבל בכדי לסיים את הלולאה. פה הוא מייצג ערך ש-ZAKIF מקבל. שם הוא נחשב לערך ממשי, כסוגו של SHIUR, פה הוא נחשב למספר שלם, כסוגו של ZAKIF.

(8) המשתנה ZAKIF ישמש כזקיף. כאשר ערכו יהיה שווה ל- SOF_NETUNIM מפסיקים את ביצוע הלולאה.

(10) הערך ההתחלתי של ZAKIF יהיה 1. למעשה כל ערך חוץ מזה שקבענו עבור SOF_NETUNIM יתאים עבור הערך ההתחלתי.

(11)-(21) אפשר להתחיל את הלולאה מיד. בתחילת הלולאה קוראים את SHIUR ו-SHAOT (בשורות (13) ו-(14)), מחשבים את השכר (בשורה 15) ומדפיסים אותו. שואלים אם יש עוד נתונים ופועלים בהתאם:

- אם ZAKIF<>SOF_NETUNIM מבצעים שוב פעם את הלולאה,
- אם ZAKIF=SOF_NETUNIM לא מבצעים את הלולאה פעם נוספת ומסיימים את התכנית.

יתרון גדול בשיטה זו הוא, שלא צריך לבחון איזה ערך איננו מתאים עבור SHIUR כדי שנוכל להשתמש בו כזקיף, אשר יסמן את סוף הנתונים. יתרון נוסף הוא שלא צריכים לחזור על READLN(SHIUR) פעמיים, פעם לפני הלולאה ופעם בסופה.

אבל למה לא להגדיר את הזקיף כמשתנה בוליאני? אם נעשה כך אפשר יהיה לפשט את התכנית עוד יותר ולשנות אותה כך:

- שורות (4) ו-(5) לא תהיינה נחוצות,
- בשורה (8) המשתנה ZAKIF יוגדר כמשתנה בוליאני ולא כמשתנה שלם,
- בשורה (10) נכתוב ZAKIF:=FALSE,
- נחליף שורה (11) ב- WHILE NOT ZAKIF DO (שמשמעותו מובנת מאליו).

הקושי הוא בכך שהתיקון המוצע לא יפעל ברוב המהדרים. בשורה (20) קוראים את הערך של ZAKIF. ברוב המהדרים הסטנדרטיים אסור לקרוא ערך בוליאני כקלט. יש דרך להתגבר על המגבלה שהוזכרה בעזרת הפונקציה הבוליאנית EOF(INPUT).

3.4 הפונקציה הבוליאנית EOF

פונקציה בוליאנית (Boolean Function) היא פונקציה שתוצאתה היא אחד מן הערכים הבוליאניים True או False. הוראות שימוש בפונקציה זו שונות במהדרים שונים. לדוגמא במחשב VAX, אם קוראים נתונים בעזרת פקודת READ ומיד לאחר הנתון האחרון מקישים בו-זמנית על המקש CONTROL ועל המקש Z (יש לבדוק אם כך הדבר במחשב שבו אתה פועל). הפעולה תגרום לפונקציה EOF(INPUT) לקבל את הערך TRUE. אם קוראים בעזרת READLN, אפשר להקיש CONTROL ו-Z בכל מקום בשורה לאחר הנתון האחרון, ללא תלות במספר הרווחים אשר מפרידים בינו לבין CONTROL–Z.

במקום לכתוב EOF(INPUT) אפשר לכתוב רק EOF. ההנחה היא שמקור הקלט הוא לוח המקשים ולכן אין צורך לכתוב את המלה INPUT. הנחה זו היא ברירת מחדל (Default Option). כאשר הקלט מגיע ממקור אחר יש לפרט זאת בסוגרים לאחר המלה EOF. אפשרות זו נדונה בפרק 13, בדיון על קבצי תמליל.

התכנית הבאה משתמשת בפונקציה EOF.

```
PROGRAM MASKORET6 (INPUT, OUTPUT);
     (* התכנית הזאת מחשבת משכורת של עובדים במפעל.
                  לא צריך לדעת מראש כמה עובדים יש. *)
VAR
   SHIUR, SHAOT, SCHAR: REAL;
BEGIN
   WHILE NOT EOF DO
      BEGIN
          WRITELN ('מהו שיעור השכר לשעה? כמה שעות עבד?');
          READ (SHIUR, SHAOT);
          SCHAR := SHIUR * SHAOT;
          WRITELN (SCHAR:8:2, 'מגיע לו שכר של^^',
                   'עבור^^', SHAOT:6:1, '^שעות עבודה')
      END
END.
```

<u>הסבר:</u>

(3)-(1) השם והתיאור של התכנית.

(5)-(4) הגדרת המשתנים. שים לב, שעכשיו לא נחוץ להגדיר את המשתנים SOF_NETUNIM ואת ZAKIF כקבועים או כמשתנים בוליאניים. הסיבה לכך היא שהפונקציה הבוליאנית מוגדרת במהדר של פסקל ולכן שורות (4), (5), (8), (10), (18), (19) ו-(20) שבתכנית הקודמת מיותרות עכשיו.

(6) בזמן התחלת התכנית ערכו של EOF הוא FALSE ולכן הערך של התנאי NOT EOF הוא TRUE. בתכנית הקודמת הודענו למחשב להפסיק את הלולאה כאשר המשתנה ZAKIF מקבל ערך 0 (שורות 18 עד 20). בתוכנית זו המחשב יגלה שאין יותר נתונים כאשר הוא יתקל בצירוף CONTROL-Z יחד עם קריאת הנתון האחרון. המחשב יפעיל את הפונקציה הבוליאנית EOF (ז"א ייחס לפונקציה ערך TRUE), אשר תגרום להפסקת ביצוע הלולאה.

3.5 שיקולים לבחירה של הוראת לולאה

יש לשקול בכל מקרה אם להשתמש בלולאה WHILE..DO או בלולאה FOR..DO. המבנה FOR..DO מתאים יותר למקרה שבו ידוע מראש מספר הלולאות שרוצים לבצע. המבנה WHILE..DO מתאים למקרה של קריאת נתונים ללא ידיעה מראש כמה לולאות יש לבצע. במהלך הביצוע תנאי מסוים הופך להיות FALSE ושינוי זה גורם להפסקה של ביצוע הלולאה.

דוגמא של השימוש המתאים בשתי פקודות אלו ניתנת בתכנית הבאה:

```
PROGRAM TZIYUNIM (INPUT, OUTPUT);
                 (* בתכנית הבאה מחשבים עבור כל סטודנט
                 את הממוצע של הציונים שלו בחמישה מבחנים.
                 מדפיסים את הממוצע ולידו גרף המורכב מכוכבים. *)
VAR
     MISPAR_TALMID, SCHUM, MIKTZOA, SHURA_REIKA,
     TZIYUN, KOCHAV, MEMUZA: INTEGER;
BEGIN
   WHILE NOT EOF DO
      BEGIN
         PAGE (OUTPUT);
         WRITELN ('הכנס בבקשה מספר תלמיד ');
         READ (MISPAR_TALMID);
         WRITELN ('הכנס בבקשה 5 ציונים');
         SCHUM := 0;
         FOR MIKTZOA := 1 TO 5 DO
            BEGIN
              READ (TZIYUN);  (*הציון הוא מספר בין 0 ל-100*)
              SCHUM := SCHUM + TZIYUN
            END;
         MEMUZA := ROUND (SCHUM/5);
```

```
(22)          FOR SHURA_REIKA := 1 TO 2 DO
(23)              WRITELN;
(24)          WRITE (MISPAR_TALMID, '^':15, MEMUZA:10, '^');
(25)          FOR KOCHAV := 1 TO ROUND (0.2 * MEMUZA) DO
(26)                WRITE ( '*' );
(27)          FOR SHURA_REIKA := 1 TO 3 DO
(28)             WRITELN
(29)       END
(30)  END.
```

הרצה לדוגמא:

```
הכנס בבקשה מספר תלמיד
1111
הכנס בבקשה 5 ציונים
80    70    90    80    80
                                  שתי  שורות
                                     ריקות
^^^^^^1111^^^^^^^^^^^^^^^^^^^^^^^80^****************
|תוים 10 ||   תווים 15  ||תוים 10 |1|  כוכביות 16  |
```

<u>הסבר:</u>

(1)-(4) שם ותיאור התכנית.

(5)-(7) משתנים שנשתמש בהם בתכנית.

(9)-(29) החלק הביצועי של התכנית נמצא בתוך לולאה. נפסיק לבצע את הלולאה הזו לאחר שנסיים את הטיפול בתלמיד האחרון כאשר EOF יהיה שווה ל-TRUE.

(11) ההוראה PAGE(OUTPUT) תגרום לקידום <u>דף</u> (Page) אם יחידת הפלט המיוחסת למלה OUTPUT היא מדפסת, או להצגה של מסך נקי אם היחידה המיוחסת למלה OUTPUT היא מסוף.

(12)-(13) קריאה של מספר תלמיד.

(15) חשוב מאוד לאפס את SCHUM בכל פעם שמבצעים את הלולאה, כדי שהוא לא יכיל את הסכום המצטבר של כל הלולאות הקודמות.

(16)-(20) קריאה וסיכום של 5 הציונים של הסטודנט.

(21) חישוב הממוצע. ההוראה ROUND מבטיחה שהתוצאה תהיה מספר שלם, אחרת היא תהיה מספר ממשי.

(22)-(23) לולאה לשם "הדפסה" של 2 שורות ריקות בכדי להפריד בין הקלט והפלט של כל סטודנט.

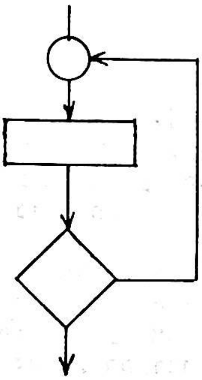
(24) שורת הפלט: מספר התלמיד ברוחב 10 תווים, רווח של 15 תווים, ממוצע הציונים ברוחב של 10 תווים ועוד רווח אחד. אין קידום של השורה.

(26)-(25) הדפסת כוכביות באותה שורה. מספר הכוכביות יחסי לערך של הממוצע של אותו תלמיד. הדפסת הכוכביות נתונה לשיקול דעתו של מתכנן הדוח. ההדפסה באותה שורה נעשתה באמצעות הפקודה WRITE, המאפשרת כתיבות חוזרות ונשנות באותה שורה ללא קידום.

(28)-(27) לולאה לשם "הדפסה" של השורה הנוכחית ושל 2 שורות ריקות נוספות בכדי להפריד בין הפלט אשר מתייחס לסטודנטים שונים.

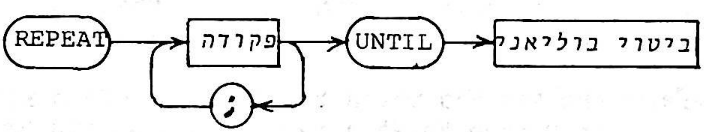
3.6 הפקודה REPEAT..UNTIL

מבנה האיטרציה השני אשר נתאר בתרשים ושנדון בו עתה הוא מבנה DOUNTIL:



המבנה השלם מורכב ממחבר, תהליך רגיל או תהליך מוגדר מראש, החלטה וחזרה למחבר. ההבדל בין DOWHILE לבין DOUNTIL הוא בכך שבמבנה החדש עוברים בתהליך ומבצעים את הפעולה המוגדרת בו לפחות פעם אחת. רק אז בודקים את התנאי שנמצא במבנה ההחלטה ועל סמך תוצאות הבדיקה מחליטים אם לחזור למחבר.

הפקודה המקבילה בפסקל מתוארת בתרשים הבא:



קיימים שני הבדלים בין לולאת WHILE..DO לבין לולאת REPEAT..UNTIL:

א. במבנה REPEAT..UNTIL מבצעים את הבדיקה רק לאחר ביצוע הפקודה (או פקודות), כיאות למבנה DOUNTIL. מאידך, במבנה WHILE..DO הבדיקה קודמת לביצוע.

ב. במבנה REPEAT..UNTIL הלולאה ממשיכה להתבצע כל עוד הערך של הביטוי הבוליאני הוא FALSE והיא מפסיקה כאשר ערך הביטוי הופך ל-TRUE. במבנה WHILE..DO המצב הפוך. הלולאה מתבצעת רק כאשר הערך של הביטוי הבוליאני הוא TRUE והיא מפסיקה להתבצע כאשר ערכו הופך ל-FALSE.

לפעמים כוונת התכנית ברורה יותר כאשר משתמשים במבנה REPEAT..UNTIL. נדגים זאת ע"י החלפה של הפקודות (6) עד (13) שבתכנית MASKORET6 בפקודות הבאות:

```
 (6)  REPEAT
 (7)       WRITELN ('מהו שיעור השכר לשעה? כמה שעות עבד?');
 (8)       READLN (SHIUR, SHAOT);
 (9)       SCHAR := SHIUR * SHAOT;
(10)       WRITELN (SCHAR:8:2, ' ^^מגיע לו שכר של',
(11)            'שעות של עבודה^', SHAOT:6:1, ' ^עבור')
(12)  UNTIL EOF
```

הסבר:

אחרי ההוראה בשורה (6) לא צריכים BEGIN ולפני השורה (12) לא צריכים END. בפקודה WHILE..DO מלים אלו שימשו סימנים לתחילה וסוף הקטע. במבנה הלולאה REPEAT..UNTIL המלה REPEAT תוחמת את הקטע בתחילתו והמלה UNTIL תוחמת אותו בסופו ולכן לא צריכים מלות עזר נוספות.

זוג ההוראות REPEAT..UNTIL EOF מדגיש בבהירות יתר ובצורה חיובית שנרצה לחזור על הלולאה עד שנגיע לסוף הנתונים. ההוראה WHILE NOT EOF DO מבטאת את אותו התנאי בצורה שלילית: "בצע את הלולאה כל עוד לא הגעת לסוף הנתונים". לכן יש להעדיף במקרה זה את הפקודה REPEAT.

כאשר משתמשים ב-"WHILE NOT EOF" וגם כאשר משתמשים ב-REPEAT..UNTIL EOF יש צורך להניח שיש לפחות נתון אחד. הסיבה לכך היא העובדה שהצירוף CONTROL-Z איננו ערך קביל עבור משתנה שלם או ממשי ותפקידו רק לסמן סיום קריאה של משתנה ואת סוף הקלט. כדי לעשות זאת, צריך להיות בקלט לפחות ערך קביל אחד, אשר CONTROL-Z יסמן את סיום הקריאה שלו ושל כל הקלט.

3.6.1 האלגוריתם של אוקלידס

נניח שיש למצוא את הגורם המשותף הגדול ביותר של שני מספרים - GCF (Greatest Common Factor). כדי לעשות זאת נסתמך על האלגוריתם של אוקלידס שאומר:

א. אם X הוא ה-GCF של A ו-B, אז ההפרש בין שני המספרים מתחלק ב-X.
ב. ה-GCF של שני מספרים זהים שווה לאותו מספר עצמו.

הוכחה ל-א' נובעת מן העובדה, שאם X הוא ה-GCF של A ו-B זה מחייב שיהיו שני מספרים A_1 ו-B_1 כך ש: $A=XA_1$ וגם $B=XB_1$ ומכאן נקבל: $B-A=X(B_1-A_1)$. המשמעות היא שההפרש B-A מתחלק ב-X. טענה ב' מובנת מאליה.

השיטה המוצעת בכדי לחשב את ה-GCF היא תהליך חוזר של חישוב ההפרש שבין שני המספרים הנתונים והחלפת הגדול שביניהם ע"י ההפרש, עד ששני המספרים שווים. לדוגמא, נתונים המספרים A ו-B. אם למשל A>B, נחליף את A בהפרש A-B. נשווה

את B אל הערך החדש של A ונמשיך להחליף את הערך הגדול בהפרש שבין המספר הגדול לבין המספר הקטן, עד שנגיע למצב שבו שני הערכים שווים. כעת קבלנו את GCF של זוג המספרים המקוריים A ו-B.

נתכנת עתה את האלגוריתם של אוקלידס.

```
 (1)  PROGRAM EUCLID (INPUT, OUTPUT);
 (2)  VAR
 (3)       A,B,GCF: INTEGER;
 (4)  BEGIN
 (5)       WRITELN ('כתוב שני מספרים שיש למצוא את ה-GCF שלהם');
 (6)       READ (A,B);
 (7)       REPEAT
 (8)           WHILE A>B DO
 (9)              A := A-B;
(10)           WHILE B>A DO
(11)              B := B-A
(12)       UNTIL  A = B;
(13)              GCF := A;
(14)       WRITELN (GCF, 'ה-GCF שווה ל:')
(15)  END.
```

<u>הסבר:</u>

(1) שם התכנית.

(2) משתני הקלט הם A ו-B ומשתנה הפלט הוא GCF.

(4)-(6) קריאה של נתוני הקלט.

(8)-(9) מחסרים B מ-A כל עוד A גדול יותר.

(10)-(11) מחסרים A מ-B כל עוד B גדול יותר.

מהלך הביצוע של התכנית:

נניח שנתון A=98, B=217. כאשר B>A, מבצעים שורה (11) והערך החדש של B יהיה 217-98=119 (A נשאר 98). B עדיין גדול מ-A ולכן שוב פעם מבצעים את (11) ומקבלים B=119-98=21 (A נשאר 98). עכשיו A גדול יותר ולכן מבצעים את (9) ומקבלים A=98-21=77 (B נשאר 21). ממשיכים ומקבלים A=77-21=56, אח"כ A=56-21=35 ואח"כ מקבלים A=35-21=14 (הערך של B נשאר 21). עכשיו A קטן מ-B ולכן מחשבים B חדש: B=B-A=21-14=7 (הערך של A נשאר 14). עכשיו A יותר גדול ולכן מחשבים את A=A-B=14-7=7 וזהו ערכו של B. בשלב זה פקודה (12) נכנסת לתוקף, מסיימים את הלולאה ומדפיסים את התוצאה בשורה (14).

<u>3.6.2 המספר הגדול ביותר במחשב</u> (*)

השלם הגדול ביותר שאפשר להחזיק במחשב שיש לו מלה בגודל m סיביות הוא $2^{m-1}-1$ (ראה נספח). נניח שלא ידוע מה ערכו של m ומעוניינים לברר זאת. על בסיס העובדה ש: $MAXINT=2^{m-1}-1$ נוכל לכתוב את התכנית הבאה:

```
(1)  PROGRAM MILAT_MACHSHEV (OUTPUT);
(2)  VAR
(3)     N, K, GODEL_MILA, MAX: INTEGER;
(4)  BEGIN
(5)      GODEL_MILA := 2;
(6)      N := 1;
(7)      REPEAT
(8)         N := 2 * N;
(9)         K := N-1;
(10)        GODEL_MILA := GODEL_MILA + 1;
(11)        MAX := N+K
(12)     UNTIL MAX := MAXINT;
(13)     WRITELN ('המספר הגדול ביותר שאפשר להחזיק במחשב זה');
(14)     WRITELN ('הגודל של מלת המחשב^', MAXINT, '^^ הוא');
(15)     WRITELN ( GODEL_MILA:20, '^^הוא')
(16)  END.
```

הרצה לדוגמא:

```
המספר הגדול ביותר שאפשר להחזיק במחשב זה
הגודל של מלת המחשב^^^32767^^הוא
^^^^^^^^^^^^^^^^^^16^^הוא
```

<u>הסבר:</u>

(1) שם התכנית. התכנית לא מקבלת קלט כלשהו ולכן אפשר לכתוב בסוגריים את המלה OUTPUT בלבד.

(2)-(3) במשתנה N נרשום את הערך העשרוני הגדול ביותר של חזקה של 2 שהמחשב יכול להחזיק. GODEL_MILA הוא משתנה שמחזיק את מספר הסיביות של המלה של המחשב אשר ערכה שווה ל-MAX.

(5)-(6) במצב ההתחלתי N=1 ו- GODEL_MILA=2. כלומר, נתונה מלה בת שתי סיביות, שבה סיבית אחת מיועדת לסימן וסיבית אחת מיועדת למספר. הערך הגדול ביותר (שהוא גם חזקה של 2) שאפשר להחזיק במלה זו הוא 1.

(8)-(11) כופלים את N ב-2. הערך הגדול ביותר הוא לא מכפלה של 2 אלא אחד קטן ממנו, לפי הנוסחה $2^{m-1}-1$. נוסיף K=N-1 ל-N בכדי לקבל את MAX. עתה נבדוק אם ערך זה של MAX אמנם שווה ל-MAXINT. אם התשובה היא "כן" הגענו לגודל של מלת המחשב. אחרי הביצוע הראשון של הלולאה, הערך N משתנה מ-1 ל-2, K שווה ל-1 ואח"כ MAX=3, GODEL_MILA=3. ואמנם, 3 הוא הערך הגדול ביותר שאפשר להחזיק במלה של 3 סיביות, כאשר הסיבית השמאלית היא סיבית הסימן.

אי אפשר לכתוב MAX:=N+N-1, כי אם MAX באמת שווה ל-MAXINT, כאשר נחבר N+N תיווצר גלישה ואי אפשר יהיה לחסר את הספרה 1. לכן פועלים בשני שלבים: תחילה מחשבים K=N-1 ורק אח"כ מחברים את K ל-N. אפשר גם לכתוב MAX:=N+(N-1). התוצאה תהיה זהה, כי מחסרים לפני שמחברים ולא מגיעים למצב של גלישה.

3.7 תרגילים

3.7.1 שאלות

(1) מה יהיה הערך של S בסוף התכנית שרשומה להלן:

```
PROGRAM LOOPS (INPUT, OUTPUT);
VAR     T, S: INTEGER;
BEGIN
        I := 1;
        S := 0;
        REPEAT
           S := S+1;
           I := I+1
        UNTIL I= 100;

        FOR I := 1 TO 100 DO
           S := S-1
END.
```

(2) נתונים תכנית ונתוני קלט. מהו הפלט שלה?

```
PROGRAM WHILE_DO (INPUT, OUTPUT);
VAR
     A, B, C, D: INTEGER;
BEGIN
     A := 0;
     B := 1;
     C := 0;
     D := 17;
     WHILE A<19 DO
         BEGIN
             READ (D);
             A := A+D;
             WRITE (C, D);
             C := C + B
         END;
     WRITE (C)
END.
```

נתוני הקלט הם: 1 2 3 2 4 5 9 6 4 8 9

(3) מהו הפלט של התכנית הבאה:

```
PROGRAM REPEAT_UNTIL (OUTPUT);
CONST
     A := -1;
VAR
     B, C, D, E:INTEGER;
BEGIN
     E := 0;
     B := 1;
```

```
        C := 0;
        REPEAT
           E := E+A;
           D := C+B;
           B := C;
           C := D;
           WRITE (D)
        UNTIL D > 20;
        WRITE (E)
     END.
```

(4) מהו הערך של MISHTANEH שיודפס בסוף כל קטע? תוכל להניח שהמשתנים מייצגים ערכים שלמים.

א.

```
MISHTANEH := -5;
FOR INDEX := 5 DOWNTO -10 DO
    MISHTANEH := MISHTANEH + 1;
WRITELN (MISHTANEH);
```

ב.

```
MISHTANEH := -10;
FOR INDEX1 := 6 TO 10 DO
  FOR INDEX2 := 7 TO 12 DO
    MISHTANE := MISHTANE + INDEX1 + INDEX2;
WRITELN (MISHTANE);
```

ג.

```
MISHTANE := 4;
WHILE MISHTANE >= 0 DO
  BEGIN
    MISHTANE := MISHTANE + 2;
    WHILE MISHTANE >= -2 DO
       MISHTANE: = MISHTANE - 3
  END;
WRITELN (MISHTANE);.
```

ד.

```
MISHTANE := -2;
REPEAT
   MISHTANE := MISHTANE + 1;
   WHILE MISHTANE > -2 DO
        MISHTANE := MISHTANE - 2
UNTIL MISHTANE >= -2;
WRITELN (MISHTANE);
```

3.7.2 תכניות

(1) הדפס טבלה שתכיל את N, N**2, שורש N והסכום של כל המספרים השלמים מ-1 עד N עבור 1<=N<=25. הפלט צריך להראות כך:

N	N**2	SHORESH (N)	1 AD N
1	1	1	1
2	4	1.414	3
:	:	:	:

(2) כתוב תכנית שמקבלת כקלט את המספר N ומדפיסה N! (N עצרת).

(3) אב חסך 2,500 שקלים עבור חינוך אוניברסיטאי לבתו. הבת יכולה להרוויח 200 שקלים לחודש. שכר לימוד הוא 100 ש' לחודש והוצאות האש"ל הן 350 ש' לחודש. הכסף מופקד בבנק ומושכים אותו עבור הוצאות בתחילת כל חודש. הבנק נותן ריבית של 2% בסוף כל חודש. כתוב תכנית שתדפיס דו"ח שיפרט כמה כסף יש בחשבון בסוף כל חודש, עד לניצול כל הקרן שהופקדה. מבנה הדו"ח:

חודש	הכנסה	הוצאות	יתרה בסוף חודש לפני ריבית	ריבית	יתרה בסוף חודש אחרי ריבית
1	200.-	450.-	2,250.-	50.-	2,300.-
2	200.-	450.-	2,050.-	46.-	2,096.-
:	:	:	:	:	:

(4) כתוב את התכנית שכתבת בשאלה 3 והוסף את האפשרות לקבל קלט ידני של ההכנסות וההוצאות החודשיות, במקום להשתמש בערכים קבועים מראש (200, 100 ו-350).

(5) עליך להרחיב כמה מן התכניות שכתבת בפרק 1 (מתכנית 4 עד תכנית 21) כדי שתפעלנה לא רק על קבוצה אחת של נתונים, אלא על מספר רב של קבוצות. כתוב את התכניות לשני מצבים, כאשר אתה יודע את מספר הקבוצות וכאשר אינך יודע זאת.

(6) אב עשיר רוצה ללמד את בתו לנהל את כספה. הוא נותן לה 150$ אבל דורש שתכין רשימה של הפריטים שהיא רוצה לקנות. ברשימה צריך להופיע מספר הדגם של הפריט ומחירו. הבת מחליטה להשתמש במיקרו-מחשב שלה בכדי לבדוק אם הכסף שעומד לרשותה יספיק לה לקניותיה. היא מזינה כקלט את מספרי הפריטים ואת מחיריהם. יתכן שהסכום הכולל גבוה מ-150$.

התכנית תדפיס ברשימה את הפריטים לפי סדר הקלט, כל עוד המחיר הכולל שלהם אינו עולה על 150$. בסוף הרשימה יודפס סכום הכסף שהבת מבקשת להוציא, אם ערך זה קטן או שווה ל-150$. אם הסכום גבוה יותר, התכנית תדפיס רק את הפריטים שהמחיר הכולל שלהם נמוך מ-150$. בסוף הרשימה יודפס הסכום שאושר והסכום שהנערה רצתה לקבל לשם קניות, אם הוא גבוה מהסכום שהקציב לה אביה.

(7) רוצים לבדוק אם מספר N הוא ראשוני. השיטה הפשוטה ביותר היא לחלק את N בכל המספרים אשר קטנים מ-N או שווים ל-N. אם נחלק את N בכל אחד מהם ותמיד נקבל שארית, נוכל לומר ש-N הוא מספר ראשוני. הפלט יראה כך:

המספר N הוא ראשוני: XXXXX

כאשר N מייצג את הערך הנבדק ו-XXXXX יוחלף ע"י המלה TRUE או FALSE.
רמז: מגדירים משתנה בוליאני ששווה לערך N MOD I ובדוק את ערכו בעבור הערכים של I אשר קטנים או שווים לערך N DIV 2.

(8) כתוב תכנית שמשתמשת בשיטת ניוטון בכדי להוציא שורש. השיטה מתוארת כך: אם A הוא אומדן של השורש של X, יהיה (A+(X/A))/2 אומדן יותר טוב. אם הדיוק D נתון (למשל:0.001), יש להמשיך בתהליך עד שמקבלים $DX > |X-A^2|$. הדפס את הערך של A ושל DX.

(9) בהמשך לתרגיל הקודם, הצג טבלה של מספר הצעדים שנדרשו בכדי להגיע לדיוק הנכון עבור כל ערך של D מ-0.1 עד 0.001, כאשר יורדים בצעדים שגודלם 0.001 .

(10) נתון לוח דיקט בעל המימדים הבאים: רוחב: 1.5 מטרים.
אורך: 2 מטרים.
עובי: 0.05 מטרים.

בכדי להגיע לערימת לוחות דיקט בגובה של 10מ', עליך לפעול כך:
- חתוך לרוחב את לוחות הדיקט שבערימה. בראשונה לרשותך לוח אחד.
- שים בערימה את גזרי הלוחות זה על גבי זה.

שים לב שבכל שלב אתה מכפיל את מספר גזרי הדיקטים.

כתוב תכנית שתתאר את התהליך ותציג אותו בטבלה. לדוגמא,

שלב	רוחב הדיקט	אורך הדיקט	גובה הדיקט
0	1.5	2	.05
1	0.75	2	.10
2	0.375	2	.20
3	0.175	2	.40

(11) כתוב תכנית שמוצאת את השלם הגדול ביותר שהריבוע שלו קטן מ-142,555.

(12) נתונים A ו-B. כתוב תכנית שמוצאת את הערך N הגדול ביותר אשר יקיים את התנאי: $A^N<B$. למשל, אם נתונים A=3, B=10 אז N=2, כי $3^2<10$ ואם N=3 אז $3^3>10$ ולכן התשובה היא N=2.

(13) כתוב תכנית שמוצאת את הסכום של N האיברים הראשונים בסדרות האלו:

א. $1 + 3 + 5 + 7 + \dots + 2N-1$

(*) ב. $1 + 2X + 3X^2 + 5X^4 + \dots + NX^{N-1}$

(*) ג. $X + \frac{-1}{3}X^3 + \frac{1}{5}X^5 + \frac{-1}{7}X^7 + \dots + \frac{(-1)^{N-1}}{(2N-1)}X^{2N-1}$

(14) אדם קונה דירה ומקבל משכנתא. התשלום החודשי R עבור המשכנתא מוגדר ע"י הנוסחה הבאה:

$$R = \frac{Pi\ (1+i)^N}{(1+i)^N - 1}$$

כאשר:

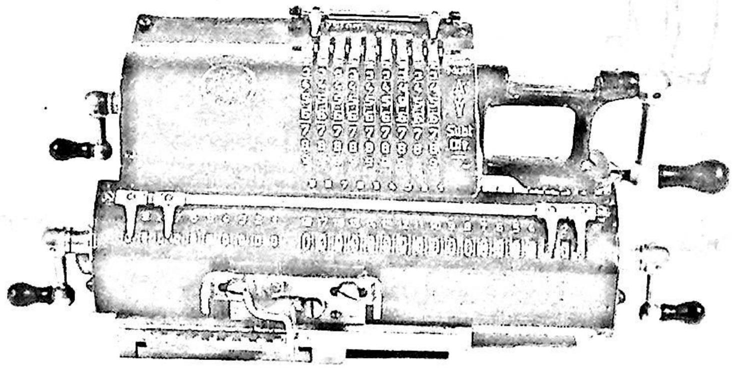
P שווה לקרן (עלות הבית).

i שווה לשיעור הריבית החודשית מובע במספר עשרוני (למשל 0.06).

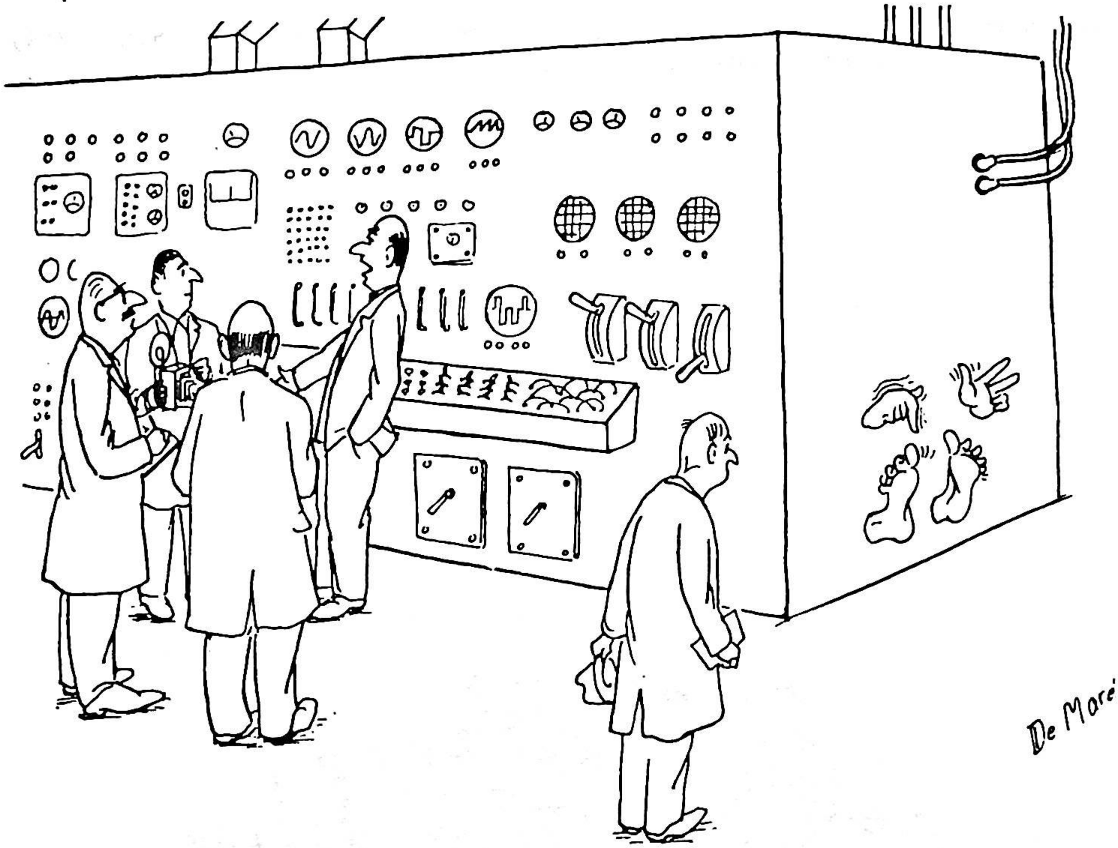
N שווה למספר החודשים לתשלום המשכנתא.

כתוב תכנית בכדי לחשב את התשלום החודשי עבור המשכנתא. נתונים הערכים של P, i, N.

(15) כתוב תכנית בכדי לחשב את מספר הספרות בצד שמאל של הנקודה העשרונית במספר ממשי שהוא הקלט. רמז: חלק ב-10 עד שהמנה יהיה פחות מ-1.



מכונת חישוב מכנית



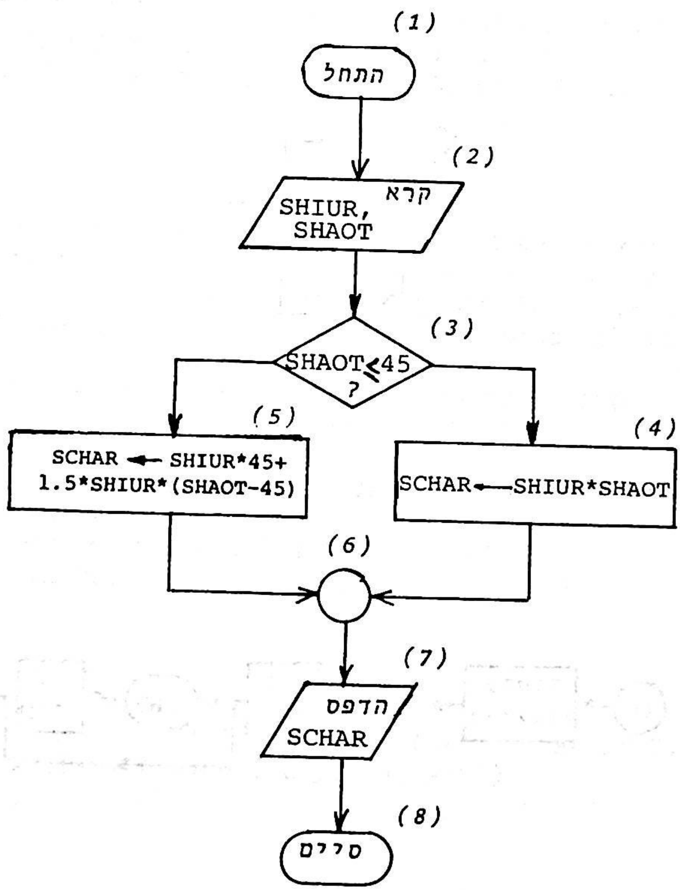
"רבותי, פעולת המחשב מבוססת על השיטה החשבונית המקובלת"

פרק 4

הסתעפויות

4.1 הפקודה IF

עד עתה למדנו על כתיבת תכניות שבהן הפעולות בוצעו לפי סדר הכתיבה של הפקודות, מהפקודה הראשונה ועד הפקודה האחרונה. לפעמים רוצים לבצע פעולה אחת כאשר תנאי מסוים מתקיים ופעולה אחרת אם התנאי לא מתקיים. למשל, כאשר מחשבים את המשכורת של עובד במפעל יש לחשב בדרך אחת אם הוא עבד שעות



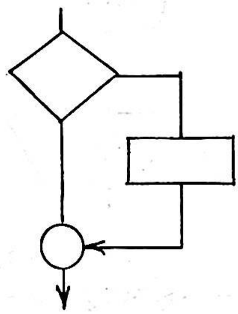
רגילות בלבד ולחשב בדרך אחרת את משכורתו אם עבד שעות נוספות. את האפשרות של "הדרך האחרת" צריך לקחת בחשבון מראש, בעת עיצוב התכנית.

תרשים הזרימה הקודם מציג טיפול במקרה כזה, באמצעות המבנה IFTHENELSE.

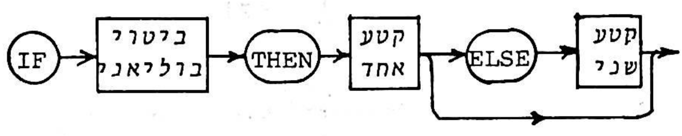
המבנה IFTHENELSE ("אם אז אחרת") מורכב מסמלים (3) עד (6). זהו אחד משני מבנים של הסתעפות מותנית (Conditional Branching), או בחירה (Selection). כבמקרים אחרים, אפשר להחליף את הסמלים (4) ו-(5) בסמלים המייצגים קטע.

התרשים הזה הינו הרחבה של הבעיה לחישוב משכורת שכבר עסקנו בה קודם. בסמל מספר (3) בודקים אם הפועל עבד במשך השבוע 45 שעות או פחות. אם התשובה היא חיובית, הוא אינו זכאי לשכר מיוחד לשעות נוספות ולכן עוברים (מסתעפים) לסמל (4) ומחשבים את שכרו לפי השיטה הרגילה. במקרה זה לא מבצעים כמובן, את סמל (5). אם כתוצאה מהבדיקה מוצאים שהפועל עבד יותר מ-45 שעות, מחשבים את משכורתו לפי השיטה החדשה: את 45 השעות הבסיסיות כופלים בשיעור השכר הרגיל לשעה ואת עודף השעות (שעות נוספות=SHAOT-45) כופלים ב-1.5 של השכר הרגיל, כי הפועל מקבל תמורה של 150% מהשכר הרגיל עבור עבודה בשעות נוספות. במקרה זה לא נבצע את סמל (4) אלא את סמל (5). בכל מקרה, מגיעים למחבר (6) וממשיכים.

מבנה שונה במקצת מן המבנה שדנו בו עתה, הוא המבנה IFTHEN ("אם אז"). מבנה זה מתאים למקרה שבו תוצאה אחת של הבדיקה מחייבת פעולה המתוארת בסמל מלבן (מלבן רגיל או מלבן המייצג קטע), אבל תוצאה אחרת אינה מחייבת פעולה כלשהיא. המבנה נראה כך:



נוכל עתה לתאר את הפקודה בשפת פסקל, אשר מתאימה למבנים השונים שאפשר ליצור בעזרת הפקודה IF:



אם לביטוי הבוליאני יש ערך TRUE בזמן ביצוע פקודה זו, נבצע את הקטע הראשון ("אחד") ולא את הקטע השני. אם לביטוי הבוליאני יש ערך FALSE, קיימות שתי אפשרויות:

א. מבנה IFTHEN: אין המשך לפקודה זו וממשיכים בביצוע התכנית.
ב. מבנה IFTHENELSE: הפקודה ממשיכה אל המלה ELSE שאחריה יש קטע לביצוע, למקרה שערך הביטוי הוא FALSE.

נכתוב עתה תכנית על פי תרשים הזרימה ונשתמש בפקודה IF.

```
 (1)  PROGRAM MASKORET7 (INPUT, OUTPUT);
 (2)                 (* התכנית מחשבת משכורת של פועל.
 (3)            לוקחים בחשבון אפשרות של שעות נוספות. *)
 (4)  CONST
 (5)    SHAOT_R = 45;                  (* שעות רגילות *)
 (6)  VAR
 (7)    SHIUR, SHAOT, SCHAR: REAL;
 (8)  BEGIN
 (9)        WRITELN ('מהו השכר לשעה? כמה שעות עבד?');
(10)        READ (SHIUR, SHAOT);
(11)        IF SHAOT <= SHAOT_R THEN
(12)           SCHAR := SHIUR * SHAOT
(13)        ELSE
(14)           SCHAR := SHIUR * SHAOT_R +
(15)                       1.5 * SHIUR * (SHAOT - SHAOT_R);
(16)        WRITELN (SCHAR:8:2, 'מגיע לו שכר^^');
(17)        WRITELN ('שעות עבודה^', SHAOT:6:1, '^^עבור')
(18)  END.
```

<u>הסבר:</u>

(15)-(11) בדוגמא זו יש פקודה אחת בלבד בקטע הראשון שבשורה (12) ובקטע השני בשורות (14) עד (15). לשם בהירות ורהיטות הקריאה של התוכנית הוצגו המלים IF ו-ELSE כשהן "מיושרות" לשמאל זו מעל זו. גם הקטע הראשון והשני "מיושרים", כאשר שניהם מוזזים ימינה למלים IF ו-ELSE.

נניח שרוצים להדפיס הודעה על כך שהפועל עבד, או לא עבד שעות נוספות. לשם כך יש צורך להחליף את השורות (11) עד (15) בקטע התכנות הבא:

```
IF SHAOT <= SHAOT_R THEN
    BEGIN
       SCHAR := SHIUR * SHAOT;
       WRITELN  ('הוא לא עבד שעות נוספות')
    END
ELSE
    BEGIN
       SCHAR:= SHIUR * SHAOT_R +
                     1.5 * SHIUR * (SHAOT - SHAOT_R);
       WRITELN ('הוא עבד שעות נוספות')
    END;
```

בדוגמא זו רוצים לבצע יותר מפקודה אחת בכל אחד מהסעיפים של התנאי ולכן יש לחבוק את הפקודות האלו במפרידים BEGIN ו-END, לאחר המלים THEN ו-ELSE. כותבים את המלים BEGIN ו-END בשורות נפרדות, כשהן מוזזות קצת ימינה מ-IF ו-ELSE ואת הפקודות כותבים בהזזה נוספת לימין, לשם בהירות ההצגה של התוכנית. כללי הכתיבה של התכנית הם המלצה בלבד ואת פרטיהם נלמד בהמשך.

4.1.1 פקודות IF משולבות

הקטע הראשון וגם הקטע השני בפקודת IF שבתכנית MASKORET7 יכולים להכיל כל פקודה שהיא וגם פקודת IF. למשל, נניח שיש שתי רמות של שעות נוספות. פועל אשר עובד למעלה מ-45 שעות מקבל 150% עבור 10 השעות הנוספות הראשונות (הרמה הראשונה) ו-200% עבור כל שעה מעבר ל-55 השעות (הרמה השניה). נגדיר קבוע שיהיה שווה למספר השעות הרגילות שהעובד צפוי לעבוד (SHAOT_R=45) וקבוע נוסף שיהיה שווה לסוף הרמה הראשונה של שעות נוספות (OMES=55). כדי ליישם מדיניות תשלום זו יש להחליף בתכנית את השורות (11) עד (15) ולכתוב את הפקודות הבאות:

```
(1)  IF SHAOT <= SHAOT_R THEN
(2)     SCHAR := SHIUR*SHAOT
(3)  ELSE
(4)     IF SHAOT <= OMES THEN
(5)        SCHAR := SHIUR*SHAOT_R + 1.5*SHIUR * (SHAOT-SHAOT_R)
(6)     ELSE
(7)        SCHAR := SHIUR*SHAOT_R + 1.5*SHIUR*10 +
(8)                     2.0*SHIUR * (SHAOT-OMES);
```

בדוגמא זו, הקטע השני לאחר ELSE כולל פקודת IF נוספת. פקודת IF כזו נקראת <u>פקודת IF משורשרת</u> (Chained IF).

אם SHAOT<=SHAOT_R מבצעים את שורה (2) ומדלגים להמשך התכנית, לפקודה שלאחר שורה (8). אם SHAOT>SHAOT_R, ממשיכים בשורות (3) ו-(4). אם SHAOT<=OMES מבצעים את שורה (5) ומדלגים לפקודה שאחרי שורה (8). אם, לעומת זאת, SHAOT>OMES מבצעים את שורות (7)-(8) וממשיכים בפקודה הבאה.

בקטע הבא מוצגת דרך נוספת לתכנות של התנאי הזה:

```
(1)  IF SHAOT > SHAOT_R THEN
(2)     IF SHAOT > OMES THEN
(3)        SCHAR := SHIUR*SHAOT + 1.5*SHIUR*10 +
(4)                          2*SHIUR*(SHAOT-OMES)
(5)     ELSE
(6)        SCHAR := SHIUR*SHAOT_R + 1.5*SHIUR*(SHAOT-SHAOT_R)
(7)  ELSE
(8)     SCHAR := SHIUR*SHAOT;
```

בדוגמא זו, הקטע הראשון לאחר THEN כולל פקודת IF נוספת. פקודה זו נקראת <u>פקודת IF מקוננת</u> (Nested IF), מלשון "קן".

בקטע תכנות זה, אם מתברר אחרי הבדיקה בשורה (1) שאמנם SHAOT<=SHAOT_R, עוברים לשורה (8), אחרת ממשיכים עם הבדיקה בשורה (2). אם SHAOT>OMES

מבצעים את שורה (3) ומדלגים לפקודה שאחרי שורה (8). לשורה (5) מגיעים אם SHAOT<=OMES, אך גם SHAOT>SHAOT_R, ז"א SHAOT_R<SHAOT<=OMES. במקרה זה מבצעים את שורה (6) ומדלגים לפקודה שאחרי שורה (8).

בשני קטעי התכנות הקודמים, שבהם השתמשנו בפקודות IF, המלים IF ו-ELSE הופיעו פעמיים:

- בקטע הראשון מוצגת פקודת IF משורשרת: IF..ELSE..IF..ELSE.
- בקטע השני מוצגת פקודת IF מקוננת: IF..IF..ELSE..ELSE.

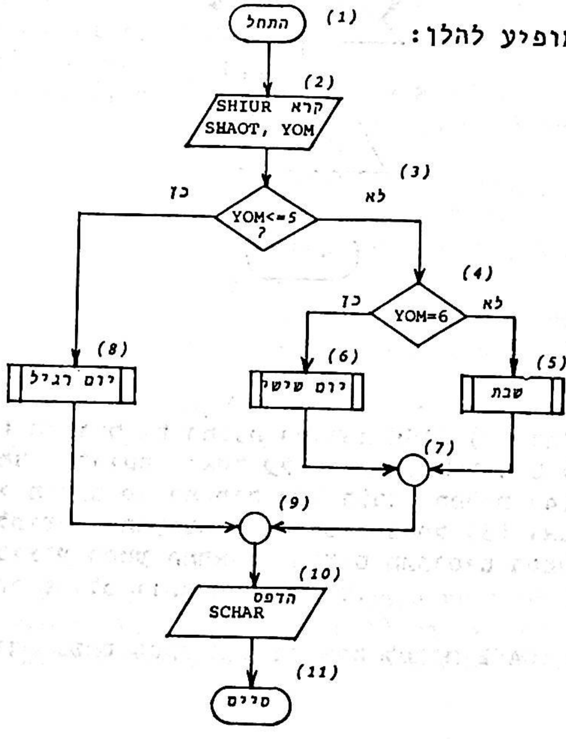
כאשר אפשר לייחס ELSE ליותר מ-IF אחד, הכלל הוא ש-ELSE תמיד מיוחס ל-IF אשר קרוב אליו ואשר עדיין פנוי. תמיד מתחילים לייחס את המלים הפנימיות ביותר קודם. לכן, בקטע השני ה-IF וה-ELSE הפנימיות שייכות זו לזו וה-IF וה-ELSE החיצוניות שייכות זו לזו. זאת אומרת, ה-ELSE בשורה (7) מיוחסת ל-IF בשורה (1) וה-ELSE בשורה (5) מיוחסת ל-IF בשורה (2). יש לשים לב לכך ששני הזוגות של IF..ELSE (הפנימיות והחיצוניות) מיושרות בהתאם. בכל מקרה, לא צורת היישור והכתיבה קובעות את משמעות הפקודה, אלא הכלל שהוסבר קודם.

4.2 הפקודה CASE

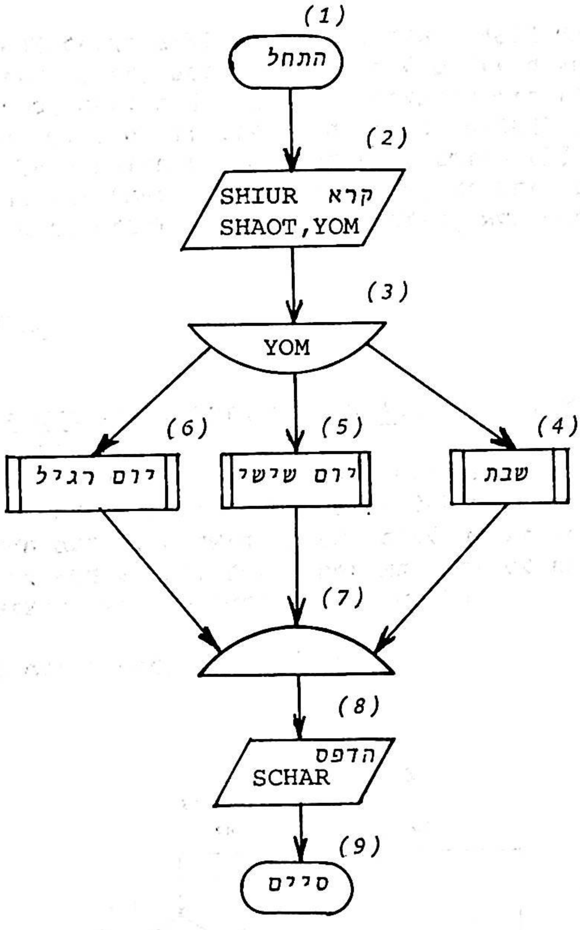
אפשר להציג הסתעפות מותנית באמצעות מבנה CASE. זהו המבנה היחיד בשיטת התכנות המבני, אשר מתיר יותר משתי יציאות.

נניח שהשכר שפועל מרוויח עבור שעות נוספות תלוי ביום בשבוע. בימים א'-ה' הוא מקבל 150% מהשכר הרגיל עבור כל שעה מעבר ל-8 שעות, ביום ו' הוא מקבל 200% עבור כל שעה מעבר ל-5 שעות. בשבת, כולל מוצאי שבת, הוא מקבל 300% מהשכר הרגיל עבור כל שעת עבודה. בכדי לחשב את שכרו של הפועל ליום מסוים, דרוש משתנה נוסף שיציין את יום העבודה (מ-1 עד 7).

תרשים זרימה מתאים מופיע להלן:

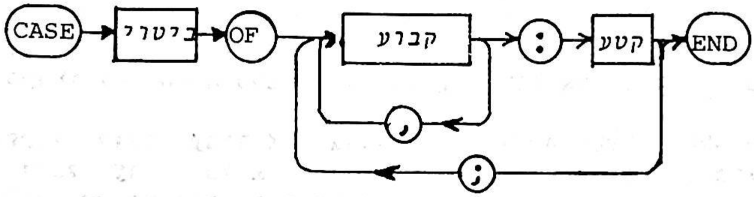


מכיון שצריך לטפל בשלושה מקרים נפרדים (יום א-ה, יום ו ויום שבת), דרושים שני מבני החלטה, (3) ו-(4). אילו היו למשל, 10 מקרים, היינו זקוקים ל-9 מבני החלטה. מכיון שתרשים כזה עלול להיות מסורבל, נהוג להשתמש במבנה מיוחד, אשר מוצג בתרשים הבא:



סמלים (3) עד (7) מהווים את המבנה המורכב CASE ("מקרה"). בסמל (3) כותבים את הביטוי שעומד לבדיקה ואשר על פי ערכו מחליטים לאיזה כיוון להמשיך. בתרשים זה הביטוי מורכב מן המשתנה YOM בלבד. סמלים (4)-(6) הם הקטעים אשר יתבצעו בהתאם לתוצאת ההערכה של הביטוי בסמל (3) ואשר רק אחד מהם יתבצע בכל מקרה. אחרי ביצוע הקטע המתאים, החצים מתכנסים בסמל (7), שהוא הסמל היחיד בתכנות מבני שקולט יותר משני חצים.

נתאר עתה את הפקודה בשפת פסקל אשר מקבילה למבנה CASE:



הביטוי אשר נבדק בפקודה CASE חייב לייצג ערך שלם. הוא יכול להיות משתנה, כמו בתכנית זו, או שהוא יכול להיות ביטוי אריתמטי. הקבועים, אשר מייצגים את טווח הערכים שהביטוי יכול לקבל, נקראים תוויות מקרה (Case Labels). הקטע הבא אחרי הנקודתיים מתבצע כאשר הביטוי שווה לאחת מתוויות המקרה שנמצאות לפני הנקודתיים.

משתמשים בפקודה CASE כאשר בוחנים משתנים שמקבלים יותר משני ערכים. בפקודה IF משתמשים כאשר בוחנים משתנים שיכולים לקבל שני ערכים בלבד.

נכתוב תכנית על פי תרשים הזרימה לחישוב שכר יומי ונשתמש בפקודת CASE. אם הקטע מכיל יותר מפקודה אחת, נצטרך להקיף את הפקודות ב-BEGIN וב-END, כמו שעושים בתכנות פקודת IF.

```
 (1)  PROGRAM MASKORET8 (INPUT, OUTPUT);
 (2)            (* התכנית הזאת מחשבת משכורת יומית
 (3)     של פועל כאשר תנאי התשלום תלויים ביום בשבוע *)
 (4)  VAR
 (5)            SHIUR, SHAOT, SCHAR: REAL;
 (6)            YOM: INTEGER;
 (7)  BEGIN
 (8)     WRITELN ('SHIUR, SHAOT, YOM :הקש לפי הסדר הזה');
 (9)     READ (SHIUR, SHAOT, YOM);
(10)     CASE YOM OF
(11)       1, 2, 3, 4, 5: IF SHAOT <= 8 THEN
(12)                         SCHAR := SHIUR * SHAOT
(13)                      ELSE
(14)                         SCHAR := SHIUR*8+SHIUR*1.5*(SHAOT-8);
(15)                   6: IF SHAOT <= 5 THEN
(16)                         SCHAR := SHIUR*SHAOT
(17)                      ELSE
(18)                         SCHAR := SHIUR*5+SHIUR*2*(SHAOT-5);
(19)                   7:    SCHAR := SHIUR*3*SHAOT
(20)     END;
(21)     WRITELN  ( SCHAR:8:2, 'השכר הוא^^');
(22)     WRITELN (YOM:3, 'שעות עבודה ביום^', SHAOT:6:1 ,'עבור^^')
(23)  END.
```

<u>הסבר:</u>

המשתנה YOM יכול לקבל את הערכים 1 עד 7. אם הערך של YOM משתנה מ-1 עד 5, המחשב יבצע את השורות (11) עד (14) ויעבור לשורה (21). שורה (11) מדגימה שיותר מתוית מקרה אחת יכולה להיות קשורה לאותו קטע.

אם הערך של YOM הוא 6, המחשב יבצע את השורות (15)-(18) ויעבור מיד לשורה (21). אם הערך של YOM הוא 7, המחשב יבצע שורה (19) ולא את השורות שקדמו לה. המלה END בשורה (20) מסמנת את הסיום של פקודת CASE.

אם הערך שהתכנית קולטת עבור YOM גדול מ-7 כתוצאה מטעות בקלט למשל, המשתנה SCHAR לא יקבל ערך, כי אין תוית מקרה עבור אפשרות זו. כדי להמנע ממצב כזה, אפשר לתקן את התכנית, כדי שתתריע על שגיאה:

```
IF (YOM>0) AND (YOM<8) THEN
  : : :            (* כאן באות שורות (10) עד (20) *)
ELSE WRITELN ('הערך של יום לא חוקי');
```

ישנם מהדרים שמתירים לרשום ELSE או OTHERWISE בסוף רשימת תויות המקרה כחלק אינטגרלי של פקודות CASE. בדוגמא שלמטה נדגים אפשרות זו:

```
(1)  PROGRAM YOMHABA (INPUT, OUTPUT);
(2)                      (* בתכנית הזאת מחשבים את התאריך הלועזי
(3)                          של מחר על פי התאריך הלועזי של היום. *)
(4)  VAR
(5)    YOM ,CHODESH, SHANA, YAMIM_BACHODESH: INTEGER;
(6)    TAUT_CHODESH, TAUT_YOM: BOOLEAN;
(7)  BEGIN
(8)    TAUT_CHODESH := FALSE;
(9)    TAUT_YOM := FALSE;
(10)   WRITELN  ('הקש בבקשה תאריך: יום, חודש ושנה');
(11)   READLN  (YOM, CHODESH, SHANA);
(12)   WRITELN (YOM:1, '/', CHODESH:1, '/', SHANA:1, 'היום אחרי')
(13)   CASE CHODESH OF
(14)   1, 3, 5, 7, 8, 10, 12: BEGIN
(15)                             YAMIM_BACHODESH: = 31;
(16)                             IF YOM > 31 THEN
(17)                                TAUT_YOM := TRUE
(18)                          END;
(19)             4, 6, 9, 11: BEGIN
(20)                             YAMIM BACHODESH:= 30;
(21)                             IF YOM > 30 THEN
(22)                                TAUT_YOM:= TRUE
(23)                          END;
(24)                       2: IF ((SHANA MOD 4 = 0) AND
(25)                                (SHANA MOD 100 <> 0)) OR
(26)                                (SHANA MOD 400 = 0) THEN
(27)                             BEGIN
(28)                               YAMIM_BACHODESH:= 29;
(29)                               IF YOM > 29 THEN
(30)                                  TAUT_YOM := TRUE
(31)                             END
(32)                          ELSE
(33)                             BEGIN
(34)                               YAMIM_BACHODESH := 28;
(35)                               IF YOM > 28 THEN
```

```
                              TAUT_YOM := TRUE
                          END;
                      OTHERWISE
              (* זוהי ההרחבה שהוזכרה קודם. הסבר מלא יבוא אח"כ. *)
                      WRITELN ('לא ניתן לחשב. טעית במספר החודש');
                          TAUT_CHODESH := TRUE
                          (* כאן סוף הקטע של OTHERWISE *)
   END;
   IF TAUT_YOM THEN
      WRITELN ('לא ניתן לחשב. טעית בכתיבת יום בחודש');
   IF YOM = YAMIM_BACHODESH THEN
      BEGIN
        YOM := 1;
        IF CHODESH = 12  THEN
           BEGIN
             CHODESH := 1;
             SHANA := SHANA+1
           END
        ELSE
           CHODESH := CHODESH+1
      END
   ELSE
      YOM := YOM+1;
   IF NOT (TAUT_YOM OR TAUT_CHODESH) THEN
      WRITELN ('הוא ', YOM:1, '/', CHODESH:1, '/', SHANA:1)
END.
```

הרצות אפשריות:

א. הקש בבקשה תאריך: יום, חודש ושנה
12, 13, 1987
היום אחרי 12/13/1987
לא ניתן לחשב. טעית במספר החודש

ב. הקש בבקשה תאריך: יום, חודש ושנה
28, 2, 1985
היום אחרי 28/2/1985
הוא 1/3/85

ג. הקש בבקשה תאריך: יום, חודש ושנה
29, 2, 1983
לא ניתן לחשב. טעית בכתיבת יום בחודש

<u>הסבר:</u>

(1) בתכנית זו הקלט הוא תאריך לועזי. הפלט יכול להיות אחת משתי תשובות:

א. אם התאריך שניתן כקלט הוא תאריך חוקי, כלומר מספר החודש פחות מ-12 ומספר היום מתאים לאותו חודש, הפלט הוא התאריך של היום הבא.

ב. אם התאריך שניתן כקלט איננו תאריך חוקי, הפלט הוא הודעת טעות ותיאור של הבעיה.

(9)-(8) קביעת ערכים התחלתיים.

(11)-(10) קבלת הקלט.

(12) כאשר לא רוצים להדפיס רווחים, מציינים רוחב של 1 עבור הערך של המשתנה, למשל CHODESH:1. במקרה זה, אם המספר הוא בן שני תווים, שניהם יודפסו, אחרת יודפס תו אחד בלבד.

(18)-(14) בחודש רגיל יש 31 יום. אם YOM גדול מ-31, TAUT_YOM יהפוך להיות TRUE.

(23)-(19) טיפול מקביל בחודשים שיש בהם 30 ימים.

(31)-(24) חישוב מספר הימים בחודש פברואר בשנת עיבור (החודש בן 29 יום). בדרך כלל כל שנה שמתחלקת ב-4 היא שנת עיבור, פרט למקרה של מספר שנה שמתחלק ב-100. גם כאן יש יוצא מן הכלל: שנה שמתחלקת ב-400 היא בכל זאת שנת עיבור.

(37)-(33) חישוב מספר הימים בחודש פברואר בשנה שאינה שנת עיבור.

(42)-(38) המלה OTHERWISE אינה סטנדרטית בפסקל ולכן אינה קיימת בכל הגירסאות של השפה. השימוש בה מפשט את התכנות של מבנה CASE. בלי אפשרות זו, היינו צריכים לבדוק את המקרה של CHODESH>12 ע"י פקודת IF נפרדת. בקטע הזה מדפיסים הודעת טעות כאשר מספר החודש איננו מספר שלם בין 1 ל-12. אפשר גם לכתוב בקטע זה מספר פקודות, מבלי להשתמש ב-BEGIN וב-END. התחום נקבע ע"י המלים OTHERWISE ו-END.

(45)-(44) הודעה על טעות בהזנה של מספר היום בחודש.

(56)-(46) אם הקלט הוא היום האחרון בחודש, אז היום הבא חייב להיות היום הראשון בחודש הבא, ראה שורות (48) ו-(55). אם הקלט הוא גם היום האחרון בחודש וגם החודש האחרון בשנה, אז החודש הבא הוא החודש הראשון בשנה ומספר השנה עולה ב-1, ראה שורות (52)-(51).

(58)-(57) פקודת ELSE זו הינה בת זוג של פקודת IF שבשורה (46). אם תאריך הקלט הוא לא היום האחרון בחודש, מוסיפים אחד ליום הקלט כדי לקבל את תאריך הפלט.

(60)-(59) הדפסה של תאריך הפלט כאשר לא היתה טעות בתאריך הקלט.

4.3 הפקודה GOTO

נניח שהתשלום עבור עבודה בשעות נוספות לאישה עם ילדים קטנים גבוה יותר מאשר התשלום הרגיל. במקום 150%, 200% ו-300% עבור תשלום שעות נוספות ביום רגיל, ביום שישי ובמוצאי שבת בהתאמה, שיעורי התשלום לאישה הם 175%, 225% ו-325%, בהתאמה. איך נוכל להכניס פרט זה לתכנית MASKORET8 אשר כתבנו קודם?

הפקודה CASE אינה מתאימה לפתרון בעיה זו, כי יש בה מנגנון כדי לטפל בערכים שונים של משתנה אחד בלבד. בדוגמא זו יש שני משתנים: הראשון מציין אם לפנינו "עובד רגיל" או "אישה עם ילדים קטנים" והשני מציין את היום בשבוע. כל תוית מקרה תצטרך להיות מורכבת משני ערכים בהתאמה, כאשר הראשון יציין מצב משפחתי והשני יציין את מספר היום בשבוע. תויות מקרה כאלו, המצביעות על הערכים של שני משתנים, אינן חוקיות בשפת פסקל ולכן אי אפשר להשתמש כאן בפקודת CASE.

יש לפחות שתי דרכים לתכנת בעיה זאת. הדרך הראשונה היא להשאיר את מבנה CASE כמו שהוא ולהוסיף פקודת IF לכל מקרה ומקרה. נניח שהערך של המשתנה הבוליאני YELADIM הוא TRUE בעבור אישה עם ילדים קטנים, וערכו FALSE בכל מקרה אחר. נתכנת לדוגמא את הפתרון, אשר מתאים לתוית מקרה 7:

```
7:   IF YELADIM THEN
          SCHAR := SHIUR*3.25*SHAOT
     ELSE
          SCHAR := SHIUR*3*SHAOT;
```

החסרון בפתרון זה הוא שלמעשה יש 6 מקרים שונים. שלושת המקרים היסודיים התלויים ביום: YOM<=5, YOM=6 ו-YOM=7 ושלושה מקרים נוספים בכדי לטפל בעובדת שיש לה ילדים קטנים. התכנית מבליטה רק 3 מקרים, אשר קשורים למשתנה YOM. ההתייחסות למשתנה YELADIM חבויה בתוך הקטע שקשור לכל תוית מקרה. אילו היו 5 משתנים היינו זקוקים ל-5 רמות של פקודות CASE, או של פקודות IF מקוננות זו בתוך זו. קינון יקשה מאוד בהבנת התכנית.

דרך אחרת כדי לפתור בעיה זו היא ע"י שימוש בפקודת דילוג (Branching Instruction), או סיעוף שנכנה אותה GOTO ושהמבנה שלה הוא:



הפקודה GOTO מבצעת הסתעפות בלתי מותנית (Unconditional Branching). כאשר מגיעים אליה בזמן ביצוע, התכנית מפסיקה את רצף הפקודות ומדלגת תמיד לפקודה מסוימת, אשר התוית (Label) שלה רשומה בפקודת GOTO. בזה היא שונה מן הפקודות שגרמו להסתעפות מותנית (IF, CASE), אשר אינן מעבירות תמיד את הפיקוח אל מקום קבוע בתכנית.

נכתוב את התכנית לטיפול בתנאי הבעיה החדשים.

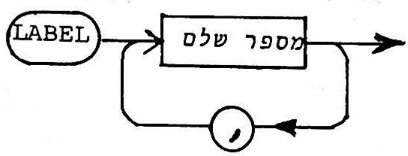
```
PROGRAM MASKORET9 (INPUT, OUTPUT);
              (* התכנית הזאת מחשבת משכורת יומית של פועל.
               מתחשבים ביום בשבוע וגם במצב משפחתי. *)
LABEL
              99,1000;
VAR
              SHIUR, SHAOT, SCHAR: REAL;
              YOM, CHILDREN      : INTEGER;
```

```
                YELADIM               : BOOLEAN;
BEGIN
   READ (SHIUR, SHAOT, YOM, CHILDREN);
   IF   ( YOM<1) OR (YOM>7 )
    THEN
     BEGIN
        WRITELN ('יש טעות בנתונים');
        GOTO 1000                                  (* צא מן התכנית *)
     END;
   IF  CHILDREN=0 THEN
        YELADIM := FALSE
   ELSE
     IF CHILDREN=1 THEN
        YELADIM := TRUE
     ELSE
        GOTO 1000;                                 (* צא מן התכנית *)
   IF YELADIM AND (YOM<=5) THEN
       BEGIN
          IF SHAOT<=8 THEN
              SCHAR := SHIUR*SHAOT
          ELSE
              SCHAR:=SHIUR*8+SHIUR*1.75*(SHAOT-8);
          GOTO 99                        (* דלג אל נקודת המפגש *)
       END;
   IF (NOT YELADIM) AND (YOM<=5) THEN
       BEGIN
          IF SHAOT<=8 THEN
              SCHAR := SHIUR*SHAOT
          ELSE
              SCHAR := SHIUR*8+SHIUR*1.5*(SHAOT-8);
          GOTO 99                          (* דלג לנקודת המפגש *)
       END;
   IF YELADIM AND (YOM=6) THEN
          BEGIN
             IF SHAOT<=5 THEN
                 SCHAR := SHIUR*SHAOT
             ELSE
                 SCHAR := SHIUR*5+SHIUR*2.25*(SHAOT-5);
             GOTO 99                       (* דלג לנקודת המפגש *)
          END;
   IF (NOT YELADIM) AND (YOM=6) THEN
              BEGIN
                 IF SHAOT<=5 THEN
                     SCHAR := SHIUR*SHAOT
                 ELSE
                     SCHAR := SHIUR*5+SHIUR*2*(SHAOT-5);
                 GOTO 99                   (* דלג לנקודת המפגש *)
              END;
   IF YELADIM AND (YOM=7) THEN
          BEGIN
```

```
(59)            SCHAR := SHIUR*3.25*SHAOT;
(60)            GOTO 99              (* דלג לנקודת המפגש *)
(61)          END;
(62)        IF (NOT YELADIM) AND (YOM=7) THEN
(63)          SCHAR := SHIUR*3*SHAOT;
(64)  99:  ;                         (* נקודת מפגש, מחבר *)
(65)        WRITELN  ( SCHAR:8:2, 'השכר הוא^^',
(66)          YOM:3, 'שעות עבודה ביום^' , SHAOT:6:1, 'עבור^^');
(67)  1000: END.
```

<u>הסבר:</u>

(4) כל מספר שלם בין 1 ל-9999 יכול לשמש <u>תוית</u>. התוית משמשת ככתובת של פקודה לביצוע, כדי שנוכל לפנות אליה בפקודה GOTO.
על השימוש בתויות צריכים להצהיר <u>בהצהרת תוית</u> (Label Declaration), שמקומה מיד אחרי הצהרת התכנית. אי אפשר להשתמש במספר כתוית ללא הצהרה מראש. המבנה הכללי של ההצהרה:



ההצהרה בשורות (4)-(5) מכריזה על שתי התויות 99 ו-1000, אשר נשתמש בהן בחלק הביצועי של התכנית. נסמן בתויות אלו שתי פקודות שאליהן רוצים להסתעף. סדר הכתיבה הוא (משמאל):
תוית נקודתיים (מומלץ) רווח גוף-הפקודה

(12)-(17) בדרך כלל נמנעים מלהשתמש בפקודה GOTO. אבל יש מקרים בהם הדבר נוח. למשל, אם מנסים לאמת את הנתונים, כמו בקטע זה, אפשר לכתוב:

```
IF (YOM=0) OR (YOM=7) THEN
    WRITELN ('יש טעות בנתונים')
ELSE    (* שאר התכנית *)
```

בשיטת תכנות זו, כמעט כל התכנית כלולה ב-ELSE וזה לא נוח כאשר היא ארוכה. לכן מעדיפים להעביר את הביצוע, במקרה של טעות, לפקודה מספר 1000 בשורה (67). יש לשים לב לכך שכאשר השליטה מגיעה ל-1000 אין מה לבצע, שהרי הגענו לסוף התכנית. התוית 1000 איננה התוית של END, כי END איננה פקודה לביצוע ולכן איננה יכולה לקבל תוית. התוית 1000 הינה התוית של <u>פקודת דמה</u> (Dummy Statement), או <u>פקודה ריקה</u> (Empty Statement) שמקדימה את END.

משתמשים בפקודת דמה בשני מקרים:

א. כאשר הפקודה GOTO מעבירה את השליטה לסוף התכנית, כמו במקרה זה.
ב. כאשר יש קטעי תכנות רבים אשר מסתעפים לאותו מחבר. חשוב יותר להדגיש את התוית כנקודת כינוס, מאשר להבליט פקודה כלשהיא שקשורה לאותה תוית.

כאשר משתמשים בפקודה GOTO, כדאי לכתוב הערה שמסבירה לאן מפנים את השליטה, כפי שנעשה בשורה (16).

(24)-(18) בדיקת נתונים וקביעת ערכים בוליאניים ע"י פקודות השמה. ההערות לגבי הקטע (12)-(17) מתאימות גם פה.

(32)-(25), (40)-(33), (48)-(41), (56)-(49), (61)-(57), (63)-(62):
אלה המקרים שהיינו רוצים לפרט בעזרת פקודת CASE, אם אפשר היה להשתמש בה בתכנית זו. הפקודה GOTO מאפשרת לבנות מבנה CASE מלאכותי, כמו שמוצג בקטעים המוזכרים.

בעבור המבנה המוצג בשורות (62)-(63) אין צורך בפקודת GOTO ובהסתעפות, שהרי הפקודה הנושאת את התוית 99 נמצאת מיד אחריו.

(64) התוית 99 משמשת כנקודת מפגש ("מחבר"). בתכנית זו כתבנו אותה כפקודת דמה ולא כפקודה לביצוע. אפשר היה להצמיד את התוית 99 לשורה (65), שיש בה פקודה, מבלי להשפיע על הלוגיקה של התכנית. בחרנו לכתוב אותה בנפרד בשורה (64) עם סימן הפרדה (;), כדי להדגיש שעיקר חשיבותה כנקודת מפגש, בדומה ל-END של פקודת CASE.

(66)-(65) הדפסת הפלט.

4.4 הנחיות לשימוש ב-GOTO (*)

4.4.1 התפתחות מבני תכנות בפסקל

שפת התכנות העילית הראשונה היתה שפת <u>פורטרן</u> (FORTRAN). נציג כאן מספר מבני תכנות בסיסיים, אשר היו נהוגים בשפה זו ונדגים את הפתרונות המוצעים בשפת פסקל אשר פותחו בכדי לפשט את התכנות ולהקל על התכניתן. מן הראוי להדגיש שבניבים החדשים של פורטרן יש מבנים דומים למבנים של פסקל.

בגירסאות המקוריות של FORTRAN היו רק שלושה מבני פיקוח כדי לשנות את מסלול ההתקדמות הסדרתי של התכנית:
א. הפקודה IF..THEN (בלי ELSE).
ב. הפקודה GOTO.
ג. הפקודה DO שכללה חלק מן האפשרויות של פקודת FOR..DO בפסקל.

למעשה, ניתן להשתמש במבנים (א) ו-(ב) ובצירופיהם בכדי להציג את כל המבנים הכלולים בפסקל.

בדוגמאות שבהמשך, המלים באותיות "גדולות" הן מלים שמורות של השפה. את המלים באותיות "קטנות" יש להחליף בשמות משתנים, ביטויים, או פקודות, בהתאמה.

שם המבנה	המבנה בפסקל	תכנות בפסקל אשר דומה לפקודות פורטרן בסיסיות
IF..THEN..ELSE	IF bitui THEN pkuda1 ELSE pkuda2	IF bitui THEN GOTO 1; pkuda2; GOTO 2; 1: pkuda1; 2:
WHILE..DO	WHILE bitui DO pkuda	1: IF NOT bitui THEN GOTO 2; pkuda ; GOTO 1; 2:....
REPEAT..UNTIL	REPEAT pkuda UNTIL bitui	1: pkuda; IF NOT bitui THEN GOTO 1;

FOR..DO	FOR mishtane :=
	gvul_tachton TO gvul_elyon
	DO
	pkuda
		zmani1:= gvul_tachton;
		zmani2:= gvul_elyon;
		IF zmani1 > zmani2 THEN
		GOTO 2;
		mishtane:= zmani1;
		pkuda;
		1: IF mishtane = zmani2 THEN
		GOTO 2;
		pkuda;
		mishtane:= mishtane+1;
		GOTO 1;
		2: ....

CASE..END	CASE bitui OF erech1 : pkuda1 ; erech2 : pkuda2 ; .... END	mishtaneh:=bitui; IF mishtaneh=erech1 THEN pkuda1 ; IF mishtaneh=erech2 THEN pkuda2 ; ....

מתוך הדוגמאות שהוצגו אפשר להסיק שלא כדאי להשתמש בפקודה GOTO. התפקיד של פקודה זו כלול כבר בפקודות אחרות בעלות עוצמה, אשר כלולות בשפת פסקל ומטרתן להקל ולפשט את התכנות.

4.4.2 מבנים בפסקל לעקיפת GOTO

בדוגמא הבאה אפשר לראות את התועלת של השימוש במבנים הקיימים בשפת פסקל, לעומת מבנים מאולתרים הנבנים בעזרת פקודת GOTO.

<u>הבעיה:</u> יש למצוא את המספר הגדול מבין שלושה מספרים נתונים.

עם GOTO

```
1:  READ (A,B,C);
    IF A>=B THEN
       GOTO 2;
    IF B>=C THEN
       GOTO 3;
    GOTO 5;
2:  IF A>=C THEN
       GOTO 4;
    GOTO 5;
3:  MAX:=B;
       GOTO 6;
4:  MAX:=A;
       GOTO 6;
5:  MAX:=C;
6:  IF NOT EOF THEN
       GOTO 1;
```

מבנים של פסקל, ללא GOTO

```
REPEAT
  READ (A,B,C);
  IF (A>=B) AND (A>=C) THEN
     MAX:=A
  ELSE
     IF (B>=A) AND (B>=C) THEN
        MAX:=B
     ELSE
        MAX:=C
UNTIL EOF;
```

מומלץ להשתמש בפקודה GOTO רק במקרים הבאים:

א. כאשר אין מבנה בשפת פסקל שמתאים לדרישות התכנית. למשל, בתכנית MASKORET9 נדרש מבנה שגורם הסתעפות לתוית הנכונה על סמך ערכם של שני משתנים. אי אפשר לתכנת זאת עם מבנה CASE של פסקל.

ב. כדי לצאת ממסלול הפקודות אל קטע תכנית נפרד, כאשר מגלים מצב שגיאה חמור, אשר איננו מאפשר להמשיך בביצוע התכנית. אפשר אז לכתוב את קטע התכנות למצב השגיאה בתוך פקודת ELSE, אבל אין זה רצוי להדגיש מקרה חריג לעומת עיקר התכנית.
בשורות (12)-(17) של התכנית MASKORET9 בצענו הסתעפות לתוית של סיום התכנית במצב של שגיאה.

כאשר עובדים במסוף, אפשר להמנע משימוש בפקודה GOTO אם מתכנתים את הבדיקה של הקלט בעת הזנתו לתכנית:

```
YOM:= -1;
WHILE  (YOM<1) OR (YOM>7) DO
  READLN  (YOM);
```

בשיטה זו קובעים למשתנה YOM ערך התחלתי, אשר נמצא מחוץ לתחום הערכים הקבילים. אח"כ חוזרים על פקודת READ עד אשר המפעיל יזין ערך קביל למשתנה. אפשר להשתמש בפקודה REPEAT ולתכנת את הקטע הזה בדרך שונה, מבלי לקבוע ערך התחלתי למשתנה YOM:

```
REPEAT
   READLN (YOM)
UNTIL (YOM>=1) AND (YOM<=7)
```

השינוי בתנאי נובע מן העובדה שבפקודה WHILE <u>ממשיכים</u> כל עוד התנאי אמת, ובפקודה REPEAT <u>מפסיקים</u> כאשר התנאי מתאמת. לכן, התנאי בפקודה REPEAT הוא השלילה של התנאי שבפקודה WHILE. לפי חוקי די-מורגן (עיין בפרק 2) נקבל:

השלילה של: YOM<1 OR YOM>7
היא: (YOM>=1) AND (YOM<=7)

בשפת פסקל ישנן פקודות, אשר עדיפות על הפקודה GOTO. עם זאת, כאשר משתמשים בה מסיבה כלשהי, רצוי להקפיד על הכללים הבאים:
א. לא לדלג לאחור. דילוגים אלו מקשים על הקריאות של התכנית.
ב. להוסיף הערה בפקודת הסיעוף וגם ביעד הדילוג.

מחברי שפת פסקל רצו להניא את המתכנת משימוש בפקודה GOTO ולכן חייבו הצהרה מראש של תויות, הגבילו אותן למספרים טבעיים ולא התירו שימוש בשמות סימליים.

4.5 מבנה התכנית

בשלב זה הכרנו את המרכיבים השונים של שפת פסקל:
א. תווים מיוחדים כמו: - , +, /, * וכו'.
ב. תווים מיוחדים כפולים כמו: =:, <>, =>, =< וכו'.
ג. מלים שמורות, כגון AND, BEGIN, CASE וכו'.
ד. שמות מוגדרים מראש, למשל BOOLEAN, FALSE, REAL, LOG, SQRT וכו'. (שמות אלו שונים ממלים שמורות, כי אף על פי שהם מוגדרים מראש כפונקציות המוכרות ע"י המהדר, המתכנת יכול להגדיר אותם מחדש כמשתנים או כקבועים ואז הם מאבדים את משמעותם המקורית).
ה. שמות המוגדרים ע"י המשתמש: משתנים, קבועים וכו'.
ו. מספרים שלמים.
ז. מספרים ממשיים.
ח. תויות.
ט. מחרוזות, שהן סדרות של תווים המוכלים בין גרשיים.
י. מפרידים, שהם רווחים, סימן סוף שורה והערות.

אין כללים לגבי המיקום של מרכיבי השפה השונים בתכנית ובשורות הקלט המוזנות למחשב. אפשר לרשום תוית באמצע שורה, או לחלק פקודה אחת על פני מספר שורות. עם זאת, יש שני כללים לגבי מיקום המרכיבים השונים, כפי שנראה להלן. בהמשך נציין תו רווח בסימן ^ כפי שעשינו עד כה:

א. חייב להיות לפחות מפריד אחד בין מרכיבים עוקבים כאשר הם שמות, מלים שמורות, או מספרים. למשל, אם כותבים MOD10^100 בלי רווח בין ה-MOD וה-10, המהדר "מבין" שהכוונה למשתנה חדש MOD10, שהרי זהו שם חוקי למשתנה. הוא לא ידע שהכוונה היא לפעולה MOD עם מחלק 10.

ב. אסור שיהיה מפריד בתוך אחד ממרכיבי השפה, חוץ מאשר במחרוזת, כי המהדר יסיק שלפניו שני משתנים. למשל, אם מגדירים משתנה בשם MOD10 וכותבים MOD^10 המהדר יתרגם זאת כשתי מלים, כאשר הראשונה היא פעולת MOD והשניה היא המספר 10.

כאשר כותבים 6.2^E+12 והכוונה היא ל: $6.2x10^{12}$, הרווח מבטל משמעות זו, כי הוא נמצא באמצע מספר ממשי. צריך לכתוב 6.2E+12 ללא כל רווח בין התווים.

אסור לכתוב IF^MONE<^>^7^THEN כי תו רווח בין התווים של סימן היחס מחלק לשניים תו מיוחד כפול <>. הרווח שבין התו המיוחד הכפול לבין המספר 7 מותר, אבל איננו הכרחי, כי הוא נמצא בין שני מרכיבים שונים, שאינם מחייבים מפריד ביניהם.

כללי הפיסוק הם:
א. בסוף כל תוכנית, לאחר ה-END האחרון, תכתב נקודה.
ב. בין פקודה לפקודה יכתב התו נקודה-פסיק. מלים כמו END, ELSE, UNTIL אינן נחשבות כפקודות אלא כמלות עזר, או חלקי פקודות ולכן אין לכתוב נקודה-פסיק בסיום הפקודה שלפניהן.

על אף שלא כופים על הכותב תבנית כלשהי, רצוי לכתוב את התכנית לפי כללים מוסכמים. בדרך זו, גם אנשים אחרים, בנוסף על המתכנת שכתב את התכנית, יוכלו לקרוא, להבין ובמידת הצורך גם לתקן אותה. מומלץ לקיים את הכללים הבאים:

א. ליישר בצד שמאל של השורה את <u>המלים השמורות</u> הבאות:

```
PROGRAM
LABEL
CONST
VAR
BEGIN
END
```

ב. להזיז ימינה ולכתוב בשורות נפרדות את <u>BEGIN</u> ו-<u>END</u> וליישר אותן בקו אחד.

ג. להזיז ימינה ולכתוב בשורה נפרדת את <u>התוכן של ההצהרות</u>:
LABEL, CONSTANT, VAR

לדוגמא:

```
LABEL
    99;
CONST
    KOCHAV = '*';
VAR
    X, Y, Z: INTEGER;
```

ד. רצוי לכתוב רק <u>פקודה אחת בשורה</u>.

במקום לכתוב:

```
Z:=1; V:=X;
```

רצוי לכתוב:

```
Z:=1;
V:=X;
```

ה. תבנית של <u>פקודות מורכבות</u>:

```
BEGIN
    pekuda1;
    ......
    pekudaN
END;
```

ו. תבנית של הפקודה <u>IF..THEN..ELSE</u>:

```
IF tnai THEN
    pekuda1
ELSE
    pekuda2;
```

IF מיושר עם ELSE, pekuda1 מיושרת עם pekuda2.

ז. תבנית של פקודה <u>IF משורשרת</u>:

```
IF bitui1 THEN
    pekuda1
ELSE
    IF bitui2 THEN
        pekuda2
    ELSE
        IF bitui3 THEN
            pekuda3
        ELSE
            pekuda4;
```

בפקודה IF משורשרת רצוי ליישר את המלים והפקודות הבאות:

- המלה ELSE עם המלה IF הקודמת לה.
- המלה IF השנייה עם pekuda1, אשר שייכת לפקודת IF הראשונה.
- המלה IF השלישית עם pekuda2 של פקודת IF השנייה.
- pekuda3 עם pekuda4, במסגרת הזוג IF..ELSE.

ח. תבנית של הפקודה <u>IF מקוננת</u>:

```
IF  bitui1  THEN
   IF bitui2  THEN
       IF bitui3  THEN
           pekuda1
       ELSE
           pekuda2
   ELSE
       pekuda3
ELSE
   pekuda4;
```

המלים הבאות מיושרות:

- המלה ELSE הראשונה עם מלה IF השלישית (וגם עם pekuda3).
- המלה ELSE השנייה עם IF השנייה (וגם עם pekuda4).
- המלה ELSE השלישית עם IF הראשונה.

ט. תבנית עבור הפקודה <u>WHILE..DO</u>:

```
WHILE tnai DO
    BEGIN
       pekuda
    END;
```

י. תבנית עבור הפקודה <u>REPEAT..UNTIL</u>:

```
REPEAT
    pekuda1;
    :
    pekudaN
UNTIL  tnai;
```

יא. תבנית עבור המבנה <u>FOR..DO</u>:

```
FOR index:=gvul_tachton TO gvul_elyon DO
   BEGIN
      pekuda1;
      :
      pekudaN
   END;
```

יב. תבנית עבור מבנה <u>CASE..OF</u>:

```
CASE mishtane OF
     erech1: pekuda1;
     :
     erechN: pekudaN
END;
```

בדרך כלל רצוי ליישר את זוגות המלים האלה:

- BEGIN עם END,
- IF עם ELSE,
- REPEAT עם UNTIL,
- CASE עם END.

<u>הדגמה של כללי הכתיבה של תכנית פסקל:</u>

יש להדפיס את כל המספרים בסדרה של מספרים, שסימנם זהה לסימנו של המספר הראשון בסדרה.

```
PROGRAM HADPES_MISPARIM (INPUT, OUTPUT);
VAR
  MISPAR: REAL;
  SIMAN_RISHON: BOOLEAN;
BEGIN
  WRITELN ('כתוב מספרים');
  READ (MISPAR);
  SIMAN_RISHON := MISPAR>=0;  (* הסימן של המספר הראשון בסדרה *)
  REPEAT
    READ (MISPAR);
    IF SIMAN_RISHON = (MISPAR>=0) THEN
      WRITELN (MISPAR)
    ELSE
      WRITELN ('לא מתאים')
  UNTIL EOF
END.
```

הסבר:

(8) בודקים את הסימן של המספר הראשון ומחזיקים אותו במשתנה הבוליאני SIMAN_RISHON. אם המספר חיובי או אפס המשתנה SIMAN_RISHON מקבל ערך של TRUE, אחרת הוא מקבל ערך של FALSE.

(11) משווים את הסימן של המספר האחרון שקראנו עם הסימן של המספר הראשון בסדרה שהוחזק ב-SIMAN_RISHON. אם הסימנים שווים, כותבים את המספר בשורה (12), אחרת כותבים את המלים "לא מתאים" בשורה (14).

מן הראוי לשים לב לפרטים הבאים במבנה התכנית:

א. המלים PROGRAM, VAR, BEGIN ו-END מיושרות בצד שמאל של הדף.
ב. התוכן של הצהרת VAR כתוב בשורות נפרדות (3) ו-(4) בהתאם לסוגי המשתנים ומוזז ימינה.
ג. המלים BEGIN ו-END בשורות (5) ו-(16) מיושרות ומופיעות לבד בשורה.
ד. כל הפקודות בין BEGIN ו-END מוכנסות פנימה ומיושרות בקו אחד.
ה. המלה REPEAT בשורה (9) מיושרת עם המלה UNTIL בשורה (15).
ו. המלים IF ו-ELSE מיושרות בשורות (11) ו-(13) וכך גם התוכן של הפקודות האלה בשורות (12) ו-(14).

4.6 תרגילים

4.6.1 שאלות

(1) כתוב פקודה IF פשוטה, עם תנאי אחד בלבד, אשר שקולה לפקודת IF המורכבת שמוצגת כאן:

```
IF X<0 THEN
   BEGIN
      IF Y<0 THEN
         A:=X+Y+Z
   END
ELSE
   IF X=5 THEN
      A:=X+Y+Z;
```

(2) אם נתון B1=TRUE ו-B2=FALSE, מה הערך של A אחרי שתי הפקודות הבאות:

א.
```
IF B1 THEN
   IF B2 THEN A:=5
ELSE A:=10;
```

ב.
```
IF B1 THEN
   BEGIN
      IF B2 THEN A:=5
   END
ELSE A:=10;
```

(3) קרא בעיון את התכנית הבאה וענה על השאלות שבהמשך:

```
PROGRAM CHIPUS (INPUT, OUTPUT);
VAR
```

```
    MISPAR, MISPAR_BARSHIMA, I: INTEGER;
BEGIN
    READ (MISPAR);
    READ (MISPAR_BARSHIMA);
    I:=1;
    WHILE (MISPAR_BARSHIMA <> MISPAR) AND NOT EOF DO
       BEGIN
          I:=I+1;
          READ (MISPAR_BARSHIMA)
       END;
    IF MISPAR_BARSHIMA = MISPAR THEN
       WRITELN  ('המספר מופיע במקום ', I:3, 'ברשימה')
    ELSE
       WRITELN ('המספר איננו ברשימה')
END.
```

א. מה עושה התכנית?

ב. מהו הפלט עבור הקלט הבא: 1- 13 21 7 85 42 25 46 22 85 ?

(4) נתונה תכנית אשר קוראת את מספר הימים בחודש ושלושה נתוני מצב לכל יום:

- GASHUM - מציין אם ירד גשם. אם ירד גשם ערכו שווה 1.
- NAMUCH - מידת החום הנמוכה ביותר.
- GAVOA - מידת החום הגבוהה ביותר.

```
PROGRAM GESHEM (INPUT, OUTPUT);
VAR
   GASHUM, GAVOA, NAMUCH, YOM, YAMIM_BACHODESH: INTEGER;
BEGIN
    READ (YAMIM_BACHODESH);
    YOM:=0;
    GASHUM:=0;
    WHILE (GASHUM=0) AND (YOM < YAMIM_BACHODESH) DO
       BEGIN
          READ (GASHUM, GAVOA, NAMUCH);
          YOM := YOM+1
       END;
    IF GASHUM > 0  THEN
       WRITELN  ( 'יום ',YOM:5, 'היה גשום')
    ELSE
       WRITELN  ('לא היה אף יום גשום בחודש')
END.
```

קרא את התכנית בעיון וענה על השאלות הבאות:

א. מה עושה התכנית ?

ב. איך היית מרחיב את התכנית כדי שתפיק את המידע הבא:

1. היום הגשום הראשון שבו מידת החום היתה מעל ל-18^0?
2. היום הגשום הראשון שבו מידת החום היתה למטה מ- 7^0.
3. שלושת הימים הרצופים בחודש שבהם ירדה כמות הגשם הגדולה ביותר.

4.6.2 תכניות

(1) כתוב תכנית שמקבלת כקלט שני מספרים שלמים A ו-B ומוציאה כפלט משתנה בוליאני שערכו TRUE אם B מתחלק ב-A ללא שארית.

(2) בסופרמרקט מסוים נותנים הנחה של 5% על כל קנייה מעל לסכום שמוגדר כ"נקודת ההנחה". כתוב תכנית שקוראת רשימת מספרים אשר הראשון בהם מציין את נקודת ההנחה ושאר המספרים מציינים את מחיר המוצרים שניקנו. המספר אפס (0) מסמל את סוף הרשימה. ציין בפלט אם הקונה מקבל הנחה ומהו החשבון הסופי לתשלום.

נתונים לדוגמא (קרא משמאל לימין):
0, 50, 80, 60, 70, 30, 100, 60, 80, 30, 70, 60, 500.

(3) כתוב תכנית שקוראת קבוצות של 3 מספרים שמייצגים את אורכי הצלעות של משולש, שרוצים להרכיב מהם. הפלט לכל קבוצה הוא אחת מן ההודעות הבאות:

- הצלעות מהוות משולש שווה צלעות.
- הצלעות מהוות משולש שווה שוקיים.
- הצלעות מהוות משולש שונה צלעות.
- הצלעות לא יכולות להרכיב משולש (ז"א סכום האורכים של זוג כלשהוא קטן מהאורך של הצלע השלישית).

(4) קיימים נתונים מקודדים על צבע השיער וצבע העיניים של אוכלוסיה כלשהי. גודלה לא ידוע מראש ואין זקיף או דגל כדי לציין את סוף הקלט.

נתוני הזיהוי הם:

צבע שיער:		צבע עיניים:	
שחור	1 -	כחול	1 -
חום	2 -	חום	2 -
ג'ינג'י	3 -	ירוק	3 -
בלונדיני	4 -		

כתוב תכנית שתציג בטבלה את גודל האוכלוסיה ואת אחוז האנשים שיש להם כל צירוף של שיער ועיניים, למשל אחוז האנשים בעלי שיער חום ועינים ירוקות. כתוב תחילה תרשים זרימה. יש להשתמש בפקודת CASE "מלאכותית" הנבנית בעזרת פקודות GOTO.

(5) כתוב תכנית שמקבלת כקלט את הספרה A ומספר שלם B. הפלט הוא משתנה בוליאני שערכו TRUE אם הספרה A היא אחת מן הספרות שמרכיבות את המספר השלם B.

(6) כתוב תכנית שמחשבת את הפתרון למשוואה הריבועית: AX^2+BX+C.
נתונים המקדמים A, B, C ומוגדר הדיסקרימיננט $D = (B^2-4AC)**0.5$
בתשובתך התיחס לשלושת המקרים הבאים:
- D>0 - יש שני שורשים ממשיים.
- D=0 - יש רק שורש אחד.
- D<0 - יש שני שורשים לא ממשיים.

(7) נתונים שלושה מספרים A, B, C. כתוב תכנית בכדי להדפיס אותם בסדר עולה.

(8) כתוב תכנית שמקבלת כקלט סדרה של זוגות מספרים. המספר הראשון הוא הציון של תלמיד בהסטוריה והמספר השני הוא הציון של אותו תלמיד במתמטיקה. הפלט יכלול שני ערכים: מספר התלמידים שקיבלו ציון יותר גבוה בהיסטוריה מאשר במתמטיקה ומספר התלמידים שקיבלו ציון יותר גבוה במתמטיקה מאשר בהסטוריה.

(9) כתוב תכנית שקוראת סדרה של ציונים ומדפיסה כפלט: ממוצע הציונים, מספר הכשלונות (ציון פחות מ-60), מספר הציונים שנכנסו לממוצע ואת הציון הגבוה ביותר והציון הנמוך ביותר בכיתה.

(10) כתוב תכנית שמקבלת כקלט את הגודל של זוית במעלות. בדוק אם גודלה הוא בין 0 ל-360 מעלות והדפס את גודל הזוית ואת הרביע שבו נמצאת הצלע העליונה שלה. למשל, זוית בת 45 מעלות נמצאת ברביע הראשון, זוית בת 290 מעלות נמצאת ברביע הרביעי. אם הזוית גדולה מ-360 מעלות יש להדפיס הודעת שגיאה.

(11) יש להרחיב את התכנית שבתרגיל הקודם כדי שתוכל לקבוע את הרביע של הצלע העליונה של הזוית עבור כל זוית, אפילו אם גודלה יותר מ-360^0.

(12) שפר את הפלט של כמה תכניות אשר הוצגו בפרקים קודמים כדי שהמלה TRUE או FALSE לא תופיע כחלק מן הפלט. יש להשתמש בפקודות IF ו-CASE:
- תרגילים לפרק 2: תכניות 1 עד 4.
- תרגילים לפרק 3: תכנית 7.

(13) בסקר שנערך בכדי לקבוע את מידת הניצול של קיבולת המכוניות הנכנסות לעיר נרשמו הנתונים הבאים עבור כל מכונית:
- מספר האנשים במכונית.
- הקיבולת המירבית של המכונית.

כתוב תכנית אשר תדפיס טבלה שתכיל את:
- מספר המכוניות עם 1, 2, 3, 4, 5, או 6 נוסעים.
- מספר המכוניות עם ניצולת של: 0-20%
20-40%
40-60%
60-80%
80-100%

ניצולת מכונית הוא היחס בין מספר הנוסעים במכונית לבין הקיבולת המירבית שלה.

(14) כתוב תכנית שמדפיסה את גילו של אדם בשנים, אם נתון תאריך הלידה והתאריך הנוכחי.

(15) (*) כתוב תכנית שמחשבת ומדפיסה את הסכום של שני מספרים שלמים בתנאי שהסכום שלהם לא גדול מהערך MAXINT ולא קטן מהערך MAXINT-. כלומר, אין מצב של גלישה. אם יש גלישה הפלט הוא הודעה שמציינת אם היתה גלישה מעל לגבול העליון או למטה מן הגבול התחתון ובאיזה שיעור. פתור את התרגיל בשתי דרכים:
א. ע"י העברה של המספרים השלמים לייצוג ממשי. ניתן לעשות זאת ע"י הכפלה ב-1.0, או ע"י הצבה של משתנה שלם לתוך משתנה ממשי.
ב. ע"י פעולה במספרים השלמים. אם A ו-B חיוביים, אסור לשאול אם

A+B>MAXINT, כי אם התשובה היא חיובית המחשב איננו יכול להחזיק את הסכום של A ו-B, שהוא עתה גדול יותר מ-MAXINT. מותר לשאול אם A>MAXINT-B, כי זהו ערך קטן יותר מהערך המקסימלי המותר במחשב.

(16) (*) עורכים סקר בכדי לבדוק את דעת הקהל בשני נושאים:
א. אם להחזיר את יהודה ושומרון.
ב. אם להעלות את המשכורות של חברי הכנסת.

כל משתתף חייב לבחור אחת מן התשובות האפשריות הבאות:
1 - מסכים באופן מלא,
2 - מסכים,
3 - לא בטוח,
4 - לא מסכים,
5 - לא מסכים בכלל.
כל משתתף צריך לציין את ההזדהות המפלגתית שלו:
L - מזדהה עם הליכוד,
M - מזדהה עם המערך,
S - מזדהה עם מפלגה אחרת.

כתוב תכנית אשר תבצע את הפעולות הבאות:
א. תספור ותדפיס את התשובות השונות עבור כל אחת משתי השאלות.
ב. תספור ותדפיס את מספר התשובות לפי ההזדהות המפלגתית.
ג. תשווה את התשובות לכל שאלה בין התומכים בליכוד לבין התומכים במערך ותדפיס את תוצאות ההשוואה.

(17) סטודנטים רוצים לדעת באיזה מקרה העלות הכללית של לימודיהם תהיה נמוכה יותר:
א. לימודים במשך 4 שנים ועבודה בקיץ.
ב. לימודים במשך 3 שנים מלאות, כולל לימודים בקיץ, במקום עבודה.

הנתונים הדרושים להחלטה:
א. שכר לימוד שנתי, ללא סמסטר קיץ (A).
ב. שכר לימוד בקיץ (A/3).
ג. עליה באחוזים בשכר הלימוד משנה לשנה (B).
ד. הכנסה בקיץ מעבודה (C).
ה. עליה באחוזים בהכנסת הקיץ משנה לשנה (B).

הקלט לתכנית כולל את הערכים A, B, C.
הפלט הוא חישוב העלות לפי כל אחת מהאפשרויות, לשיקול דעתו של כל סטודנט.

(18) כתוב תכנית שמקבלת כקלט תאריך לועזי (במאה ה-20) וקובעת באיזה יום בשבוע חל אותו תאריך. השתמש בשיטת החישוב דלהלן:

א. חלק את שתי הספרות האחרונות של השנה (A) ב-4 ושמור את המנה (B).
ב. C <--- A+B
ג. E <--- C+D, כאשר D נקבע לפי הטבלה הבאה:

מספר החודש	D	מספר החודש	D
1	1	7	0
2	4	8	3
3	4	9	6
4	0	10	1
5	2	11	4
6	5	12	6

ד. חבר יום בחודש (Y) ותקבל F<--- E+Y.

ה. יום בשבוע שווה ל: F MOD 7 כאשר יום ראשון הוא 1, יום שני הוא 2, וכו', עד יום שבת שהוא 0.

דוגמא: אם תאריך הקלט הוא 17/8/44, נקבל:

A=44	D=3	F MOD 7 = 5
B=11	E=58	
C=55	F=75	התאריך 17/8/44 חל ביום חמישי.

(19) קיימים 4 מספרים קטנים מ-500 בעלי תכונה מיוחדת. אם לוקחים כל ספרה במספר, מעלים אותה בחזקת שלוש ומחברים את התוצאות, הסכום יהיה שווה למספר המקורי. למשל: $153 = 1^3 + 5^3 + 3^3$.
מצא לפחות אחד מהמספרים האחרים בעלי תכונה זו.



מסוף גרפי (יבמ)

פרק 5

מערכים

5.1 משתנים רב-ערכיים במימד אחד

מנהל בית החרושת מודאג מכך שהפועלים מבלים הרבה בסוף השבוע ומאחרים לבוא לעבודה ביום ראשון. כדי לתעד את טענתו, הוא רוצה לראות את סכום שעות העבודה של כל הפועלים בכל אחד מן הימים המלאים בשבוע (א'-ה'). הוא יוכיח את טענתו, אם סכום שעות העבודה של כל הפועלים יחד בכל אחד מן הימים ב' עד ה' יותר גבוה מזה של יום א'.

בהמשך כתובה תכנית, שמקבלת כקלט את שעות העבודה של כל אחד מ-50 הפועלים. הקלט בעבור כל פועל מורכב מ-5 מספרים שמייצגים את מספר השעות שהוא עבד בימים א' עד ה', בהתאמה. הפלט הוא סכום השעות שכל הפועלים עבדו בכל אחד מ-5 הימים.

```
PROGRAM MASKORET9 (INPUT, OUTPUT);
 (* התכנית מחשבת את מספר השעות שהפועלים עבדו בימים א' - ה' *)
VAR
   SHAOT1, SHAOT2, SHAOT3, SHAOT4, SHAOT5, (* שעות לכל יום *)
   RISHON,SHENI,SHLISHI,RIVIEY,CHAMISHI (*סכום לכל יום*): REAL;
   I: INTEGER;
BEGIN
   RISHON:=0; SHENI:=0; SHLISHI:=0; RIVIEY:=0; CHAMISHI:=0;
   FOR I:= 1 TO 50 DO
      BEGIN
         READ (SHAOT1, SHAOT2, SHAOT3, SHAOT4, SHAOT5);
         RISHON:= RISHON + SHAOT1;
         SHENI:= SHENI + SHAOT2;
         SHLISHI:= SHLISHI + SHAOT3;
         RIVIEY:= RIVIEY + SHAOT4;
         CHAMISHI:= CHAMISHI + SHAOT5
      END;
   WRITELN (RISHON, SHENI, SHLISHI, RIVIEY, CHAMISHI)
END.
```

הסבר:

בתכנית זו יש ריבוי של משתנים וגם ריבוי של פקודות. יש הקבלה בין המשתנים SHAOT1 עד SHAOT5 המתבטאת בפקודה (11) ובהמשך. גם המשתנים RISHON עד CHAMISHI מקבילים, כפי שנראה בפקודה (8). הפקודות (12) עד (16) גם כן מקבילות, כאשר כל אחת היא כמעט העתק של קודמתה ורק נעשו בה התיקונים המתאימים בכדי לציין את היום המתאים בשבוע.

5.1.1 מערך חד-מימדי - ווקטור

ניתן לפתור את הבעיה של ריבוי המשתנים, אם ניתן שם אחד לכל קבוצה של משתנים. למשל, לקבוצה SHAOT1 עד SHAOT5 נוכל לקרוא SHAOT ולהבדיל בין משתנה למשתנה בתוך הקבוצה ע"י הוספה של מציין (Subscript) בסוגריים מרובעים. בהתאם לכך, המשתנה SHAOT הוא משתנה רב-ערכי או מערך (Array), כי הוא מורכב מ-SHAOT[1], SHAOT[2], SHAOT[3], SHAOT[4] ו-SHAOT[5].

נוכל עתה לכתוב SHAOT[I] בכדי לתאר איבר כלשהו של המערך SHAOT ולהתייחס למכלול הערכים במערך זה אם ניתן ל-I, בזה אחר זה, את תחום הערכים מ-1 עד 5. נוכל לתאר את מערך SHAOT גם כגוש של כתובות (תאים) רצופות בזיכרון של המחשב, כפי שמוצג בתרשים הבא.

נוכל עתה לכתוב SHAOT[I] בכדי לתאר איבר כלשהו של המערך SHAOT ולהתייחס למכלול הערכים במערך זה אם ניתן ל-I, בזה אחר זה, את תחום הערכים מ-1 עד 5. נוכל לתאר את מערך SHAOT גם כגוש של כתובות (תאים) רצופות בזיכרון של המחשב, כפי שמוצג בתרשים הבא.

SHAOT [1]	SHAOT [2]	SHAOT [3]	SHAOT [4]	SHAOT [5]
7	1	9	5	8

גם את המשתנים RISHON עד CHAMISHI נוכל לקבץ בקבוצה, או במערך בשם YOM. אם נשתמש במציין J כדי לתאר איבר כלשהו במערך, נוכל לכתוב: YOM[J].

לאחר ההגדרות שכתבנו, כל אחד מן השמות SHAOT ו-YOM מייצג קבוצה של ערכים, אשר סדורים בזה אחר זה במערך חד-מימדי, אשר נכנה אותו רשימה (List) או ווקטור (Vector).

נשבץ את המערכים SHAOT ו-YOM בתוכנית הקודמת, במקום המשתנים הבודדים ונקבל את התכנית המשופרת:

```
(1)  PROGRAM MASKORET10 (INPUT, OUTPUT);
(2)  (* התכנית מחשבת את מספר השעות שהפועלים עבדו בימים א' עד ה' *)
(3)  VAR
```

```
(4)        SHAOT, YOM: ARRAY [1..5] OF REAL;
(5)        I, J  : INTEGER;
(6)  BEGIN
(7)    FOR J:=1 TO 5 DO
(8)       YOM[J]:= 0;
(9)    FOR I:=1 TO 50 DO
(10)      FOR J:= 1 TO 5 DO
(11)          BEGIN
(12)             READ (SHAOT[J]);
(13)             YOM[J]: = YOM[J] + SHAOT[J]
(14)          END;
(15)   FOR J:=1 TO 5 DO
(16)      WRITELN (YOM[J])
(17) END.
```

<u>הסבר:</u>

(4) כשם שמצהירים על משתנים, כך דרוש להצהיר גם על מערכים. בהצהרת המשתנים מציינים שכל אחד מן השמות SHAOT ו-YOM הוא <u>מערך</u> (Array) בן 5 איברים מסוג REAL. טווח המציינים של כל אחד מהמערכים הוא מ-1 עד 5, כמו שנרשם בתוך הסוגריים המרובעים. מפרידים בין הערך של המציין התחתון לבין הערך של המציין העליון <u>בשתי</u> נקודות. המלה שאחרי OF מגדירה את סוג המשתנים שבמערך. במקום REAL ניתן לכתוב INTEGER או BOOLEAN. כל איברי מערך אחד צריכים להיות מאותו סוג, שהרי יש רק מלת תואר אחת (REAL במקרה זה) בעבור כולם.

(7)-(8) מאפסים את 5 משתני הסיכום של הימים על ידי שימוש במשוואה אחת: YOM[J]:=0. המציין J יקבל את כל הערכים האפשריים בתחום בעזרת פקודת FOR. גם אם היו 100 משתני סיכום, התכנית לא היתה גדלה במספר הפקודות, אלא רק בזמן הביצוע. בכדי לטפל ב-100 משתנים יש להחליף בפקודת הצהרה (שורה 4) ובפקודות הביצוע המתאימות את המספר 5 במספר 100, כדי שהלולאה תתבצע 100 פעמים במקום 5.

(12)-(13) שתי פקודות לקריאת נתונים ולסיכום, במקום הפקודות (11) עד (16) שבתכנית הקודמת. התכנית תקרא 50 סדרות של 5 ערכים עבור כל אחד מהפועלים.

<u>5.1.2 שימוש במערך חד-מימדי</u>

נציג דוגמא נוספת לשימוש במערך חד-ממדי.

נניח שמורה בודק מבחן ורוצה לתת לסטודנטים ציונים יחסיים. לציון הנמוך ביותר הוא קובע ערך 0 ולציון הגבוה ביותר הוא קובע ערך 100. כל שאר הציונים יחושבו באופן יחסי בתוך תחום זה. למשל, אם הציון הגבוה ביותר הוא 72, יקבע לסטודנט זה הציון 100. לסטודנט שקיבל את הציון הנמוך ביותר - 63, ינתן הציון 0. הסטודנטים שציוניהם בתוך התחום 63 עד 72 יקבלו ציונים שערכם יחסי למיקומם היחסי בין הציון הנמוך והגבוה. הנוסחא המתאימה לחישוב של ציון יחסי היא:

$$TZIYUN_YACHASI = 100 * \left(\frac{TZIYUN - TZIYUN_NAMUCH}{TZIYUN_GAVOA - TZIYUN_NAMUCH} \right)$$

המשתנים TZIYUN_GAVOA ו-TZIYUN_NAMUCH מייצגים את הציון המקסימלי ואת הציון המינימלי, בהתאמה, מבין כל הציונים.

נכתוב תכנית לפתרון הבעיה:

```
(1)  PROGRAM TZIYUNIM (INPUT, OUTPUT)
(2)  VAR
(3)    TZIYUN: ARRAY [1...50] OF INTEGER;
(4)    TZIYUN_GAVOA, TZIYUN_NAMUCH, TZIYUN_YACHASI, I, J: INTEGER;
(5)  BEGIN
(6)    WRITELN ('כתוב את הציונים הגולמיים');
(7)    READ  (TZIYUN[1]);
(8)    TZIUN_NAMUCH := TZIUN[1];
(9)    TZIYUN_GAVOA := TZIYUN[1];
(10)   I:=2;
(11)   REPEAT
(12)         READ (TZIYUN[I]);
(13)         IF TZIYUN[I] > TZIYUN_GAVOA  THEN
(14)             TZIYUN_GAVOA := TZIYUN[I]
(15)         ELSE
(16)             IF TZIYUN[I] < TZIYUN_NAMUCH  THEN
(17)                 TZIYUN_NAMUCH := TZIYUN[I];
(18)         I := I+1
(19)   UNTIL EOF;
(20)   WRITELN ('ציון גולמי^^^^^^^^^^ציון יחסי');
(21)   J := I-1;
(22)   FOR I:=1 TO J DO
(23)     BEGIN
(24)        TZIYUN_YACHASI := 100 * (TZIYUN[I] - TZIYUN_NAMUCH) DIV
(25)            (TZIYUN_GAVOA - TZIYUN_NAMUCH);
(26)        WRITELN (TZIYUN_YACHASI, TZIYUN[I]:20)
(27)      END
(28) END.
```

<u>הסבר:</u>

(7)-(9) קוראים את הציון הראשון וקובעים שבינתיים הוא מייצג את הציון הנמוך ביותר וגם את הציון הגבוה ביותר.

(12)-(18) קוראים את שאר הציונים ובודקים כל אחד מהם בכדי לקבוע אם הוא גבוה מן הציון אשר נקבע כגבוה ביותר עד עכשיו, או אם הוא נמוך מן המינימום הקודם.

(21) מספר הציונים שקוראים, J, קטן באחד מהערך הנוכחי של I, כי את I מקדמים בסיום הלולאה.

(24)-(25) הביטוי (TZIYUN[I] - TZIYUN_NAMUCH) הוא מספר שלם וכך גם הביטוי (TZIYUN_GAVOA-TZIYUN_NAMUCH). אסור לבצע את החילוק הבא, כי הוא תמיד יתן תוצאה של 0, כאשר המונה קטן מן המכנה:
(TZIYUN[I] - TZIYUN_NAMUCH) DIV (TZIYUN_GAVOA - TZIYUN_NAMUCH).

כדי למנוע זאת כופלים את השבר ב-100. כפל הוא באותה רמה של חילוק ולכן הכפל ב-100 שרשום משמאל מתבצע קודם ומקבלים את התוצאה הנכונה. אפשר גם לכתוב:

```
TZIYUN_YACHASI := ROUND (100 * (TZIYUN[I] - TZIYUN_NAMUCH) /
                          (TZIYUN_GAVOA - TZIYUN_NAMUCH))
```

היתרון של שיטה זו הוא בכך שמקבלים את התוצאה מעוגלת ולא מקוצצת. למשל, אם TZIYUN_NAMUCH = 60, TZIYUN_GAVOA = 90, TZIYUN[I] = 65, נקבל תוצאה 16 לפי השיטה הראשונה ולפי השיטה השנייה נקבל 17, כי ROUND(16.66)=17.

אם TZIYUN_GAVOA שווה ל-TZIYUN_NAMUCH: המכנה יהיה שווה ל-0 וזה יגרום לשגיאת ביצוע. על אף שמקרה זה נדיר, כדאי לבדוק תנאי זה לפני שנכנסים ללולאה של שורות (22)-(27).

5.2 משתנים רב-ערכיים בשני מימדים

המערכים SHAOT ו-YOM מייצגים ערכים אשר סדורים בזה אחר זה במערך חד-מימדי, במבנה של <u>רשימה</u> (List) או של <u>וקטור</u> (Vector). קיימות קבוצות של ערכים אשר מסודרים במבנה של <u>מערך דו-מימדי</u> (Two-Dimensional Array). מערך כזה הוא <u>טבלה</u> (Table), או <u>מטריצה</u> (Matrix).

5.2.1 שימוש במערך דו-מימדי

נדגים באמצעות תוכנית את השימוש במערך דו-מימדי.

במפעל מסוים קובעים את משכורות עובדי המינהלה כפונקציה של שנות הוותק בתפקיד ושל רמת ההשכלה. הנתונים ערוכים בטבלה שבהמשך ומייצגים מספר (אחוז) שבו יש לכפול את השכר הבסיסי בכדי לקבל את השכר לכל עובד.

השכלה \ שנות ותק	פחות מ-5	בין 5 ל-10	בין 10 ל-15	יותר מ-15
בלי בגרות	100	125	150	175
בגרות	125	150	175	200
תואר ראשון לפחות	150	175	200	225

התכנית שבהמשך מחשבת את המשכורת של 25 עובדי המינהלה שבמפעל.

```
(1)  PROGRAM MASKORET11 (INPUT, OUTPUT);
(2)                    (* התכנית הזאת מחשבת את השכר של עובדי המינהלה *)
(3)  VAR
(4)        SCHAR: ARRAY [1..3, 1..4] OF REAL;
(5)        I, J, MISPAR_OVED, HASKALA, VETEK: INTEGER;
```

```
(6)  BEGIN
(7)        WRITELN  ('קרא את הטבלה לחישוב משכורת');
(8)        FOR I:=1 TO 3 DO
(9)          FOR J:=1 TO 4 DO
(10)           READ (SCHAR[I,J];
(11)       FOR I:=1 TO 25 DO
(12)         BEGIN
(13)            WRITELN ('קרא מספר-עובד, השכלה, וותק');
(14)            READLN (MISPAR_OVED, HASKALA, VETEK);
(15)            WRITELN (MISPAR_OVED:1, 'השכר של עובד מספר');
(16)            WRITELN (SCHAR[HASKALA, VETEK]:10:2, 'שוה ל-^')
(17)         END
(18)  END.
```

<u>הסבר:</u>

(4) הצהרת משתנים עבור מטריצה דומה להצהרת משתנים עבור וקטור. בהצהרת המשתנים דרושים שני זוגות של מספרים בכדי להגדיר את תחום הערכים של שני המציינים. הזוג הראשון מתייחס לתחום השינוי של אינדקס השורות בטבלה והזוג השני מתייחס לתחום השינוי של אינדקס העמודות. האיברים של כל זוג מספרים מופרדים ע"י שתי נקודות סמוכות, כאשר פסיק מפריד בין זוג לזוג.

(10) מציינים איבר בשורה I ועמודה J ע"י: SCHAR[I,J]. המשתנה I בלולאה החיצונית מקבל ערכים מ-1 ל-3. כאשר I:=1, המשתנה J אשר בלולאה הפנימית משתנה מ-1 ל-4. כלומר, 4 המספרים הנקראים תחילה הם SCHAR[1,1], SCHAR[1,2], SCHAR[1,3], SCHAR[1,4], הכתובים בשורה הראשונה של הטבלה. אח"כ הערך של I בלולאה החיצונית עולה ל-2 ושוב פעם J משתנה מ-1 ל-4 בלולאה הפנימית. לבסוף, הערך של I יהיה 3 ו-J ישתנה מ-1 ל-4 במהלך הקריאה של השורה השלישית.

<u>5.2.2 כללים לשימוש במערכים</u> (*)

בתכנית MASKORET11 אפשר היה להגדיר את המערך הדו-מימדי בצורה שונה, במבנה של ARRAY מקונן:

SCHAR: ARRAY[1..3] OF ARRAY[1..4] OF REAL

משמעות הצהרה זו היא: SCHAR הוא מערך של 3 איברים, שכל אחד מהם הוא מערך של 4 איברים. כדי לפנות לאיבר (I,J) של מערך שמוצהר בצורה זו נכתוב SCHAR[I][J]. כלומר, פונים לאיבר I של SCHAR ולאיבר J בתוך איבר I.

אסור לעשות פעולות אלגבריות על מערך, אך מותר לעשותן על האיברים שלו. למשל, נתונים המערכים A ו-B אשר מוצהרים בצורה זהה, כמו לדוגמא:

A,B: ARRAY[1..5, 1..10] OF INTEGER

אסור לכתוב A+B, A-B וכו', אבל מותר לכתוב:

A[I,J]+B[K,L]   A[I,J]-B[K,L]   וכו'.

יוצא מן הכלל יחיד לכלל שאסור לעבוד עם מערך שלם היא <u>פעולת הצבה</u>. למשל,

עבור A ו-B המוצהרים קודם מותר לכתוב <u>A:=B</u> במקום הפקודות הבאות:

```
FOR I:= 1 TO 5 DO
  FOR J:= 1 TO 10 DO
    A[I,J]:= B[I,J]
```

אם SCHAR מוגדר בהצהרה של ARRAY מקונן, מותר לכתוב SCHAR[3]:=SCHAR[2]. התוצאה היא ש-4 האיברים ב-SCHAR[3] יקבלו את הערך של 4 האיברים של SCHAR[2].

במהדרים שונים הכניסו מספר הרחבות למבנה היסודי:

א. שני המבנים

```
SCHAR1: ARRAY[1..3, 1..4] OF REAL;
SCHAR2: ARRAY[1..3] OF ARRAY[1..4] OF REAL
```

נחשבים זהים. המבנה הראשון הוא רישום מקוצר של השני.

כתוצאה מזה, ההתייחסויות הבאות:

```
SCHAR [I] [J]
SCHAR [I,J]
```

גם כן נחשבות זהות ואפשר להשתמש בכל אחת מהן בלי קשר למבנה של הצהרת המערך.

ב. בנוסף לפקודת הצבה אשר פועלת על המערך כולו כגוש אחד, ניתן גם להשוות שני מערכים שלמים במשפט תנאי בוליאני אחד.
במהדר כזה מותר לכתוב

```
IF A = B  THEN
   pekuda
```

אשר זהה לקבוצת המשפטים הבאים:

```
SHAVIM:= TRUE
FOR I:=1 TO 5 DO
   FOR J:=1 TO 10 DO
     IF A[I, J] <> B[I, J]  THEN
        SHAVIM:= FALSE;
     IF SHAVIM THEN
        pekuda
```

<u>פקודות הפועלות על איברי מערך</u>

נדגים בתכנית הבאה שימוש בפקודות הפועלות על איברי מערך:

```
PROGRAM TIPUL_BEMATRIZA (INPUT, OUTPUT);
     (* בתכנית הזאת מחשבים את סכום הערכים בשורות ובעמודות *)
        (* במטריצה A ומדפיסים אותם יחד עם המטריצה המקורית. *)
CONST
     MAX_SHUROT=6;
     MAX_AMUDOT=8;
VAR
    A:ARRAY [1..MAX_SHUROT, 1..MAX_AMUDOT] OF INTEGER;
    B:ARRAY [1..MAX_SHUROT] OF INTEGER;
    C:ARRAY [1..MAX_AMUDOT] OF INTEGER;
    I, J: INTEGER;
BEGIN
                    (* מלא במטריצה A את הערכים המתאימים *)
```

```
FOR I:=1 TO MAX_SHUROT DO
  BEGIN
    WRITELN ('הכנס שורה של^', MAX_AMUDOT:2, '^ערכים');
    FOR J:=1 TO MAX_AMUDOT DO
      READ (A[I,J]);
    READLN
  END;
                              (* אפס את הווקטורים B ו-C *)
FOR I:=1 TO MAX_SHUROT DO
  B[I]:=0;
FOR J:=1 TO MAX_AMUDOT DO
  C[J]:=0;
                              (* מצא את הסכומים הדרושים *)
FOR I:=1 TO MAX_SHUROT DO
   FOR J:=1 TO MAX_AMUDOT DO
     BEGIN
       B[I]:=B[I]+A[I,J];
       C[J]:=C[J]+A[I,J]
   END;
  (* הדפס את הפלט בצורה זו: A בשמאל, B בצד ימין, C למטה *)
PAGE(OUTPUT);
FOR I:=1 TO MAX_SHUROT DO
   BEGIN
     FOR J:=1 TO MAX_AMUDOT DO
       WRITE (A[I,J]);
     WRITELN (B[I]:15)
   END;
WRITELN;
WRITELN;
FOR J:=1 TO MAX_AMUDOT DO
  WRITE (C[J])
END.
```

מהלך של הרצה לדוגמא:

```
הכנס שורה של 8 ערכים
22  33  23  45  34  23  65  43

הכנס שורה של 8 ערכים
32  45  76  85  76  76  89  23

הכנס שורה של 8 ערכים
34  43  46  54  65  63  46  66

הכנס שורה של 8 ערכים
63  54  43  65  42  31  76  33

הכנס שורה של 8 ערכים
32  54  87  66  55  87  23  54

הכנס שורה של 8 ערכים
64  38  67  90  23  54  54  72
```

צורת הפלט:

A ⟶	22	33	23	45	34	23	65	43	288
	32	45	76	85	76	76	89	23	502
	34	43	46	45	65	63	46	66	417
	63	54	43	65	42	31	76	33	407
	32	54	87	66	55	87	23	54	458
	64	38	67	90	23	54	54	72	462
									↑ B
	247	267	342	405	295	334	353	291	⟵ C

<u>הסבר:</u>

(20)-(14) מילוי המטריצה A לפי סדר שורות.

(25)-(22) איפוס הווקטורים B ו-C. יש מהדרים שקובעים ערך התחלתי 0 לאיברים של ווקטור של מספרים, אבל כדי לא לטעות מומלץ לאפס את הווקטורים האלה בצורה מפורשת.

(32)-(27) לולאה כפולה כדי לחשב את הסכום של השורות בזו אחר זו. לאחר סריקת השורה הראשונה האיבר B[1] מקבל את ערכו הסופי, אשר שווה לסכום האיברים בשורה. מספר אחד בלבד נכנס לכל אחד מהאיברים C[1], C[2].. עד C[8] בהתאמה. בכל פעם שקוראים שורה נוספת, מקבלים B[I] לאותה שורה וצוברים עוד ערך לכל C[J]. רק בסוף קריאת השורה האחרונה, הסכומים C[J] מקבלים את ערכם הסופי.

(34) פקודה לקידום הנייר במדפסת לדף חדש, לשם הדפסת הפלט.

5.3 סוגי נתונים שמוצהרים ע"י המשתמש

כל משתנה שהוגדר עד עכשיו, הוצהר כאחד מסוגי הנתונים INTEGER, REAL, או BOOLEAN. לא הוצהר סוג נתונים לערכים קבועים, מכיון שהסוג נקבע לפי צורת הכתיבה של המספר.

המשתמש יכול להגדיר סוגים אחרים של נתונים ולהצהיר שמשתנה כלשהוא יהיה שייך לאותו סוג. נציג שתי דרכים להגדרה של סוג נתונים ע"י המשתמש.

5.3.1 הגדרה של תת-תחום

המשתמש יכול להצהיר על סוג חדש של נתונים ע"י הגדרה של <u>תת-תחום</u> (Subrange). תת-תחום הוא קבוצה חלקית של סוג נתונים שהוגדר כבר.

ההגדרה TYPE CODE=1..25 קובעת למשל, שהסוג CODE מורכב מן המספרים 1 עד 25 שנבחרו מתוך סוג הנתונים INTEGER. הערך המוזכר ראשון חייב להיות קטן מן השני ומופרד ממנו בשתי נקודות סמוכות. מספר האיברים בתוך תת-תחום חייב להיות סופי ולכן אסור לבנות תת-תחום עבור מספרים ממשיים, שהרי ידוע שיש מספר אין סופי של מספרים ממשיים בין כל שני ערכים.

אם כותבים מאוחר יותר בתכנית VAR A:CODE, הכוונה היא שבמשך התכנית המשתנה A יקבל ערכים אשר מוגבלים לתחום המוזכר. אם ננסה לתת לו ערך אחר, נקבל הודעת שגיאה.

הצורה הכללית של הצהרת TYPE היא:



לפעמים מוצגת סדרת הצהרות כזו:

```
CONST    MAXIMUM=100;
TYPE     CODE=1..MAXIMUM;
VAR      A:CODE;
```

הצהרת TYPE צריכה להרשם לאחר CONST ולפני VAR, כי ההצהרות מתבססות זו על זו. שם הסוג CODE מופיע בהצהרת TYPE ושוב פעם בהצהרת VAR, כאשר תפקידו רק לקשר ביניהן.

ניתן לקצר ולהכניס את ההגדרה של CODE בהצהרת VAR בצורה זו:

```
CONST    MAXIMUM=100;
VAR      A:1..MAXIMUM;
```

השתמשנו בהגדרה של תת-תחום בתכנית MASKORET10, כאשר הגדרנו את המערך YOM:

```
VAR    SHAOT, YOM: ARRAY[1..5] OF REAL;
```

אפשר לחלק את ההצהרה הזאת לשניים ולכתוב:

```
TYPE   RANGE=1..5;
VAR    SHAOT, YOM: ARRAY[RANGE] OF REAL;
```

<u>5.3.2 מנייה</u>

משתמש יכול להצהיר על סוג חדש של נתונים ע"י <u>מנייה</u> (Enumeration), שבה מפרטים ומציינים את כל המרכיבים של הסוג החדש. סוגי נתונים המוגדרים בדרך זו מזכירים את הסוג BOOLEAN, שמכיל רק שני ערכים: TRUE ו-FALSE. אם ננסה לתת למשתנה בוליאני ערך שונה משניים אלה, נקבל הודעת שגיאה. בצורה דומה נוכל לכתוב:

```
TYPE  YOMBASHAVUA= (RISHON, SHENI, SHLISHI, RIVIEY, CHAMISHI,
                    SHISHI, SHABBAT);
```

בהצהרה זו קובעים ששבעת הערכים הרשומים שייכים לסוג חדש שנקרא YOMBASHAVUA. ההצהרה העוקבת:

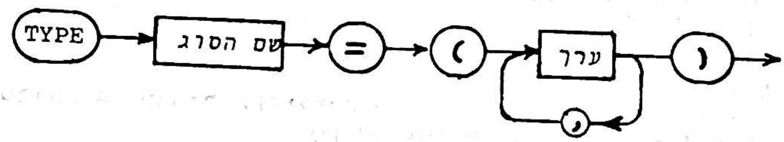
```
VAR  YOM: YOMBASHAVUA
```

קובעת, שבהמשך התכנית המשתנה YOM יוכל לקבל ע"י הצבה רק את אחד מן הערכים הכלולים בסוג YOMBASHAVUA. אם מנסים לתת לו ערך אחר מקבלים הודעת שגיאה.

אפשר לשלב את ההצהרות של הסוג והמשתנה ולכתוב:

```
VAR YOM: (RISHON, SHENI, SHLISHI, RIVIEY, CHAMISHI, SHISHI, SHABBAT)
```

נתאר בתרשים את השימוש הזה בהצהרת TYPE:



הסדר בו רושמים את הערכים קובע את יחסם ההדדי במשך התכנית. מן הדוגמא הקודמת נובעים היחסים הבאים: RISHON < SHENI <.....< SHABBAT

הערכים RISHON עד SHABBAT יכולים להופיע בכל מקום בתכנית, שם מותר לכתוב ערך שלם או בוליאני:

- בפקודת הצבה,
- בביטוי בוליאני, שנמצא בתוך אחת מן הפקודות WHILE או REPEAT,
- בהצהרת מערך, כאשר מפרטים את תחום המציינים,
- בפקודה FOR או CASE.

אי אפשר להשתמש בסוג נתונים זה בפקודות קלט/פלט.

הערכים המוגדרים בהצהרת TYPE יכולים להיות מיוחסים למשתנה מאותו סוג ע"י פקודת הצבה, אבל לא ע"י קריאה מן החוץ בעזרת פקודת READ. למשל, אסור לכתוב READ(YOM) ולספק כקלט ערך כמו SHENI, כמו שאסור לקרוא כקלט ערך עבור משתנה בוליאני. כמו כן, אי אפשר לכתוב WRITELN(YOM) ולצפות שהמחשב יציג את הערך "SHENI", אפילו אם זה הערך הנוכחי של המשתנה YOM. עם זאת, יש לזכור שאפשר לכתוב את הערך של משתנה בוליאני בעזרת פקודת WRITELN.

קיימות הרחבות של שפת פסקל, בהן ניתן לבצע פקודות קלט/פלט גם על משתנים בוליאניים וגם על משתנים המתוארים ע"י מנייה.

אפשר לכתוב הצהרת TYPE שמסתמכת על הצהרת TYPE קודמת, למשל:

```
TYPE    YOMBASHAVUA = (RISHON, SHENI, SHLISHI, RIVIEY,
                       CHAMISHI, SHISHI, SHABBAT);
        YOMALEH = RISHON..CHAMISHI;            (* יום עבודה מלא *)
```

כאשר משתמשים בסוג נתונים אשר מתואר ע"י מנייה אפשר להבין את המשמעות של המשתנים המופיעים בתכנית גם ללא הסבר נוסף, בתנאי שנותנים להם שמות מתאימים. בתכנית הבאה השתמשנו במשתנים המתוארים ע"י מנייה.

```
PROGRAM TIPUL_BEMATRIZA (INPUT, OUTPUT)
                    (* בתכנית זו מחשבים את הכמות של פירות שונים
                    שנמכרו בארץ ואת הכמויות של כל סוגי הפירות
                    שנמכרו בכל אחת מן הערים הגדולות. *)
TYPE
     SHUROT=(TAPUCHIM, TAPUZIM, SHZIFIM, ANAVIM);
     AMUDOT=(JERUSALEM, TEL_AVIV, BEER_SHEBA,
             HAIFA, ASHDOD);
VAR
```

```
(10)    A: ARRAY[SHUROT, AMUDOT] OF INTEGER;
(11)    B: ARRAY[SHUROT] OF INTEGER;
(12)    C: ARRAY[AMUDOT] OF INTEGER;
(13)    I: SHUROT;
(14)    J: AMUDOT;
(15)  BEGIN
(16)                        (* מלא מטריצה A בערכים המתאימים *)
(17)    FOR I:= TAPUCHIM TO ANAVIM DO
(18)      BEGIN        (* הסבר לפונקציה ORD ראה בסעיף 5.3.3 *)
(19)         WRITELN ('הכנס שורה של ^', ORD(ASHDOD)+1:2, '^ערכים')
(20)         FOR J:= JERUSALEM TO ASHDOD DO
(21)            READ (A[I,J]);
(22)         READLN
(23)      END;
(24)                             (* אפס את הווקטורים B ו-C *)
(25)    FOR I:= TAPUCHIM  TO ANAVIM DO
(26)        B[I]:= 0;
(27)    FOR J:= JERUSALEM TO ASHDOD DO
(28)        C[J]:= 0;
(29)                             (* מצא את הסכומים הדרושים *)
(30)    FOR I:= TAPUCHIM TO ANAVIM DO
(31)        FOR J:= JERUSALEM TO ASHDOD DO
(32)           BEGIN
(33)              B[I]:= B[I]+A[I,J];
(34)              C[J]:= C[J]+A[I,J]
(35)           END;
(36)      (* הצג את הפלט בצורה זו: A באמצע, B בצד ימין, C למטה *)
(37)    PAGE (OUTPUT);
(38)    FOR I:= TAPUCHIM TO ANAVIM DO
(39)        BEGIN
(40)            FOR J:= JERUSALEM TO ASHDOD DO
(41)                WRITE (A[I,J]);
(42)            WRITELN (B[I]:15)
(43)        END;
(44)    WRITELN;
(45)    WRITELN;
(46)       FOR J:= JERUSALEM TO ASHDOD DO
(47)       WRITE (C[J])
(48)  END.
```

<u>הסבר:</u>

(8)-(6) כותבים את שמות הערכים הקשורים עם כל אחת מן השורות והעמודות.

(14)-(10) שימוש בסוגים שהוגדרו קודם בהצהרת המשתנים.

(17), (20), (25), (27), (30), (31), (38), (40), (46) כאשר עוברים על טווח הערכים של המשתנים, מציינים את הערך המינימלי של הטווח לפי שמו (TAPUCHIM או JERUSALEM) ואת הערך המקסימלי לפי שמו (ANAVIM או ASHDOD). הפקודה מתבצעת מן הערך הקטן עד הערך הגדול.

הפלט של התכנית נשאר ללא שינוי על אף השינוי בשיטת התכנות.

5.3.3 פונקציות שניתן להפעיל על משתנים

בהמשך נציג מספר פונקציות שניתן להפעיל על משתנים מסוג המוגדר ע"י המשתמש.

א. הפונקציה ORD(n)

אפשר להשתמש בדרכים החדשות שהוצגו להצהרת נתונים ע"י המשתמש, בכדי לכתוב גירסה חדשה של התכנית MASKORET10:

```
(1)  PROGRAM MASKORET12 (INPUT, OUTPUT); (* MASKORET10 גירסה 2 של *)
(2)      (* התכנית מחשבת את מספר השעות שהפועלים עבדו בימים א'-ה' *)
(3)   TYPE
(4)       YOMALEH= (RISHON, SHENI, SHLISHI, RIVIEY, CHAMISHI);
(5)   VAR
(6)       SHAOT, YOM: ARRAY[YOMALEH] OF REAL;
(7)       J: YOMALEH;
(8)       I: INTEGER;
(9)   BEGIN
(10)    FOR J:= RISHON TO CHAMISHI DO
(11)       YOM[J]:=0;
(12)    FOR I:= 1 TO 50 DO
(13)       FOR J:= RISHON TO CHAMISHI DO
(14)          BEGIN
(15)             READ(SHAOT[J]);
(16)             YOM[J] := YOM[J]+SHAOT[J]
(17)          END;
(18)    FOR J:= RISHON TO CHAMISHI DO
(19)       WRITELN (ORD(J)+1, '^^^^^', YOM[J])
(20)  END.
```

הסבר:

(4)-(3) המשתמש מגדיר YOMALEH כסוג אשר נקבע על ידו. הוא איננו משתמש באחד מסוגי הנתונים הסטנדרטיים.

(10) המשתנה J, שסוגו YOMALEH, יכול להשתנות בתוך תחום הערכים, אשר מפורטים בהצהרת TYPE:
- בפקודה FOR...DO מותר שינוי בין ערך קטן לערך גדול,
- בפקודה FOR...DOWNTO מותר שינוי בין ערך גדול לערך קטן ממנו.

(19) אי אפשר להציג בפקודה WRITELN ערכים שמוגדרים ע"י המשתמש. למשל, אסור לכתוב WRITELN(J). עוקפים אילוץ זה באמצעות הפונקציה ORD(J), מלשון ORDinal - <u>מסודר</u>, אשר נותנת את המקום היחסי של J ברשימה שבה הוגדר. לפריט הראשון ברשימה מיוחסים ערך 0, לפריט השני מיוחסים 1 וכן הלאה עד לפריט מספר N, שמיוחסים לו ערך N-1. לכן, ORD(J)+1 יתן מספר מ-1 עד 7 שהוא מספר היום בשבוע. מספר השעות שהפועלים עבדו ביום J ניתן ע"י הערך של YOM[J].

ב. הפונקציה SUCC(n)

פונקציה נוספת שאפשר להפעיל על סוגי נתונים המוגדרים ע"י המשתמש היא SUCC(J), מלשון SUCCessor - "הבא אחרי". היא נותנת כתוצאה את האיבר שנמצא ברשימה אחרי האיבר J, למשל SUCC(RISHON)=SHENI. הביטוי SUCC(CHAMISHI) לא מוגדר, מכיון שהאיבר CHAMISHI הוא האיבר האחרון ברשימה.

בדוגמא הקודמת נוכל להחליף את השורות (18)-(19) ע"י הפקודות הבאות:

```
(1)   J:= RISHON
(2)   WHILE J<= RIVIEY DO
(3)      BEGIN
(4)         WRITELN (ORD(J)+1, '^^^^^', YOM[J]);
(5)         J:= SUCC(J)
(6)      END;
(7)   WRITELN  (ORD(J)+1, '^^^^^', YOM[J]);
```

הביטוי SUCC(CHAMISHI) לא מוגדר ולכן אי אפשר לכתוב בשורה (2): WHILE J<=CHAMISHI. אם כותבים כך, אפשר לגרום לשגיאה בביצוע הפעולה בשורה (5), כי היא תציב ב-J ערך SUCC(CHAMISHI) שאיננו מוגדר. לכן מבצעים את הלולאה רק עד שמקבלים J:=RIVIEY. בסוף הביצוע האחרון של הלולאה, SUCC(J) בשורה (5) יהיה שווה ל-CHAMISHI ובשורה (7) נבצע את ה-WRITELN עבורו.
בתכנית הקודמת, במקום השורות (1)-(7) כתבנו:

```
FOR J:= RISHON TO CHAMISHI DO
   WRITELN (ORD(J) + 1, '^^^^^', YOM[J]);
```

הערך הגבוה ביותר ש-J מקבל בפקודת FOR זו הוא CHAMISHI, כמו שאפשר ללמוד מן התיאור של הפקודה שניתן בסעיף 4.4. לכן, לא תהיה חריגה מעבר לגבולות YOMALEH שהוגדרו עבור המשתנה J.

ג. הפונקציה PRED(n)

פונקציה הפוכה ל-SUCC(J) היא PRED(J), מלשון PREDecessor - "זה שקודם". למשל, PRED(CHAMISHI)=RIVIEY. הביטוי PRED(RISHON) איננו מוגדר מכיון שהערך RISHON הוא הראשון ברשימה ואין לפניו ערך אחר. נוכל להחליף שורות (18)-(19) בתכנית MASKORET12 בקטע הבא, שיתן את אותן תוצאות, אבל בסדר הפוך:

```
(1)   J:= CHAMISHI
(2)   WHILE J<= SHENI DO
(3)      BEGIN
(4)         WRITELN (ORD(J)+1, '^^^^^', YOM[J]);
(5)         J:= PRED(J)
(6)      END;
(7)   WRITELN (ORD(J)+1, '^^^^^', YOM[J]);
```

גם בקטע תכנית זה, כמו בתכנית של הפונקציה SUCC(n), אפשר להחליף את השורות (1)-(7) בפקודות אחרות:

```
FOR J=: CHAMISHI DOWNTO RISHON DO
   WRITELN (ORD(J) + 1, '^^^^^', YOM[J]);
```

ד. הפקודה CASE

ישנם מהדרים אשר אינם מאפשרים לכתוב ערכים של משתנים המתוארים ע"י מנייה. אם יש צורך להציג ערכים אלה, ניתן להחליף את השורה (19) שבתכנית בפקודות הבאות:

```
BEGIN
   CASE  J  OF
       RISHON: WRITE('RISHON');
         ::
       CHAMISHI: WRITE('CHAMISHI')
   END;
   WRITELN (YOM [J]:10:2)
END;
```

5.4 תרגילים

5.4.1 שאלות

(1) הכן הצהרות TYPE עבור המקרים הבאים:
- ימי השבוע.
- צבעי הקשת.
- רמות השכלה: גן, עממי, תיכון, הנדסאי, אוניברסיטה.
- מינים (זכר, ונקבה).
- 12 חודשי השנה.

(2) הכן הצהרות TYPE מסוג תת-תחום עבור המקרים הבאים:
- מנקודת ההקפאה עד נקודת הרתיחה בפרנהייט.
- שנים במאה הזאת.
- ימי עבודה (תת-תחום של ימי השבוע).

(3) בהמשך כתובה תכנית אשר מקבלת קלט ומפיקה פלט. מה יהיה הפלט שלה על פי הקלט הנתון?

```
PROGRAM KADURIOT (INPUT, OUTPUT);
TYPE
    TOZAOT = 1..300;
    SACHKANIM = 0..100;
    DOCH = ARRAY[SACHKANIM] OF TOZAOT;
VAR
    MISCHAK: DOCH;
    SACHKAN: TOZAOT;
    ADAM, MISHTATEF, MAX: INTEGER;
BEGIN
    MAX:= 300;
    ADAM:= 0;
    WHILE NOT EOF DO
      BEGIN
        READ (SACHKAN);
```

```
          ADAM:= ADAM+1;
          MISCHAK[ADAM]:= SACHKAN
        END;
      FOR MISHTATEF:= 1 TO ADAM DO
        BEGIN
          IF MISCHAK[MISHTATEF] < MAX THEN
            BEGIN
              MAX:= MISCHAK[MISHTATEF];
              WRITE (MISCHAK[MISHTATEF]:1, '^':5, MISHTATEF)
            END
        END
  END.
```

נתוני הקלט הם:

185	200	216	113
98	129	141	219
85	94	113	68
95	100	120	93

(4) לפניך התכנית הבאה. קרא אותה בעיון וענה על השאלות.

```
PROGRAM SEKER (INPUT, OUTPUT);
TYPE
    YETZUR = (KELEV, CHATUL, DAG);
VAR
  CHAYAT_BAYIT: YETZUR;
  MISPAR_CHAYOT,MISPAR_KLAVIM,MISPAR_CHATULIM,MISPAR_DAGIM: INTEGER;
  KOD_CHAYOT: 1..3
BEGIN
     MISPAR_CHAYOT:=0;
     MISPAR_KLAVIM:=0;
     MISPAR_CHATULIM:=0;
     MISPAR_DAGIM:=0;
     WHILE NOT EOF   DO
         BEGIN
             READ (KOD_CHAYOT);
             MISPAR_CHAYOT:= MISPAR_CHAYOT+1;
             IF KOD-CHAYOT = 1 THEN
                CHAYAT_BAYIT:= CELEV;
             IF KOD-CHAYOT = 2  THEN
                CHAYAT_BAYIT:= CHATUL;
             IF KOD_CHAYOT:= 3 THEN
                CHAYAT_BAYIT:= DAG;
             CASE CHAYAT_BAYIT OF
                CELEV: MISPAR_KLAVIM:= MISPAR_KLAVIM+1;
                CHATUL: MISPAR_CHATULIM:= MISPAR_CHATULIM+1;
                DAG: MISPAR_DAGIM:= MISPAR_DAGIM+1
             END
         END;
```

א. איך הילת מסיים את התכנית הזאת?

ב. כלב נחשב לזקן אם גילו למעלה מ-7 שנים, חתול נחשב לזקן אם גילו למעלה מ-5, ודג נחשב לזקן אם גילו למעלה משנתיים. יש להוסיף

לתכנית את הפקודות הדרושות בכדי שתפיק גם את מספר חיות הבית המוגדרות "צעירות" ואת מספר חיות הבית המוגדרות "זקנות".

5.4.2 תכניות

(1) המכפלה הפנימית של שני וקטורים היא: $\sum_{i=1}^{N} A[i]*B[i]$

ז"א A[N]*B[N] + ... + A[2]*B[2] + A[1]*B[1].

כתוב תכנית שמקבלת כקלט את:
- המספר N, אשר מייצג את מספר האיברים של שני הווקטורים,
- N האיברים של הווקטור הראשון,
- N האיברים של הווקטור השני.

חשב והדפס את המכפלה הפנימית של שני הווקטורים, אם נתון N<100.

(2) כתוב תכנית שקוראת את הערכים בווקטור ומחליפה כל איבר בסכום של כל האיברים אשר קודמים לו.

(3) כתוב תכנית שקוראת וקטור לא מסודר של N ערכים שלמים, כאשר כמה מהם מופיעים יותר מפעם אחת. הפלט יהיה:
- אותו וקטור באותו סדר, כאשר כל ערך מופיע בדיוק פעם אחת.
- מספר M ששווה למספר האיברים בווקטור החדש.

למשל, אם הווקטור המקורי הוא: 14 9 15 8 17 8 15 7
וקטור הפלט יהיה: 14 9 17 8 15 7
והערך M יהיה 6.

(4) נתונה מטריצה במבנה ריבוע קסם (Magic Square), שבו סכום המספרים שווה בכל שורה, בכל עמודה ובכל אלכסון.

6	1	8
7	5	3
2	9	4

כתוב תכנית שמקבלת כקלט את המימדים של הריבוע ואת הערכים שלו ובודקת אם הוא ריבוע קסם. אפשר להניח שהריבוע קטן מ-10X10.

(5) ריבוע לטיני הוא מערך של N*N שמכיל מספרים, כך שמספר כלשהו לא מופיע יותר מפעם אחת בשורה או בעמודה. ריבוע זה עומד בדרישות אלו:

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

כתוב תכנית שבודקת אם מערך בגודל N*N (N<10) הוא ריבוע לטיני.

(6) האחד בינואר 1986 חל ביום רביעי. כתוב תכנית שמדפיסה לוח עבור השנה 1986+N (עבור כל N). הפלט לחודש מסוים צריך להיראות כך:

SUN	MON	TUE	WED	THU	FRI	SAT
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	10	21	22	23	24	25
26	27	28	29	30	31	

JANUARY 1986

(7) נגדיר שתי מטריצות A ו-B:

$$B=\begin{matrix} b_{11}\cdots\cdots b_{1k} \\ \vdots \quad\quad \vdots \\ b_{n1}\cdots\cdots b_{nk}\end{matrix} \qquad A=\begin{matrix} a_{11} & a_{12}\cdots\cdots a_{1n} \\ \vdots & \vdots \quad\quad \vdots \\ a_{m1} & a_{m2}\cdots\cdots a_{mn}\end{matrix}$$

א. כתוב תכנית בכדי לבדוק אם מטריצה A היא סימטרית, ז"א אם $a_{ij}=a_{ji}$ עבור $1 \le i \le n$ ועבור $1 \le j \le m$.
הערה: הגדרה זו מתאימה רק למטריצה ריבועית, ז"א למצב m=n.

ב. כתוב תכנית בכדי למצוא את המכפלה C של A ו-B, שעבורה c_{ij} מוגדר כך:

$$c_{ij} = \sum_{k=1}^{n} a_{ik}\ b_{kj}$$

(8) מוסד לגמילות חסדים מנסה למכן את חלוקת הכספים למשפחות. כתוב תכנית שתקבע את הסכום שמשפחה מקבלת על פי הטבלה הבאה. נתוני הקלט הם:

- טבלה לקביעת השכר,
- רמת ההכנסה של המשפחה,
- מספר הילדים במשפחה.

	מספר הילדים					
הכנסה	1	2	3	4	5	6 ומעלה
עד 10,000	3,000	7,000	12,000	16,000	21,000	25,000 ועוד 4,000 לכל ילד נוסף
10,000 - 19,999	2,000	5,000	7,500	9,000	11,000	13,000 ועוד 2,000/ילד נ'
20,000 - 29,999	1,000	2,800	4,800	6,500	8,000	9,500 ועוד 1,300/ילד נ'
30,000 - 39,999	1,000	1,800	3,000	4,000	5,200	6,000 ועוד 800/ילד נ'
40,000 - 49,999	800	1,500	2,500	3,200	3,800	4,500 ועוד 600/ילד נ'
50,000 - 59,999	500	800	1,500	2,100	2,600	3,400 ועוד 400/ילד נ'
למעלה מ-60,000	0	0	0	0	0	0

(9) כתוב תכנית שמקבלת כקלט נתונים על כמות הגשם הממוצעת בכל חודש ומפיקה כפלט:

- כמות גשם ממוצעת בשנה, על פי משקל שווה לכל חודש,
- החודש הגשום ביותר,
- החודש השני הגשום ביותר.
- החודש שבו ירדה כמות הגשם הקטנה ביותר,
- ההפרשים בין כמויות הגשם שירדו בכל חודש וכמות הגשם הממוצעת בכל החודשים. השתמש בהצהרת TYPE להגדרת החודשים.

(10) קיימת הנחה שיש יחס בין גילו של אדם, לבין מספר הימים שהוא חולה בשנה. במפעל מסוים נערך רישום של קבוצת עובדים, כפי שמתועד בטבלה הבאה:

גיל (X):	32	34	27	25	21	44	48	22	36	52
ימי מחלה (Y):	2	0	1	2	0	4	3	0	4	6

כתוב תכנית כדי לבדוק אם יש קשר בין המשתנים האלה ע"י חישוב מקדם המיתאם שמוגדר כך:

$$r = \frac{N \sum_{i=1}^{N} X(i) * Y(i) - \{ \sum_{i=1}^{N} X(i) \} \{ \sum_{i=1}^{N} Y(i) \}}{\sqrt{\{ N [\sum_{i=1}^{N} X(i)^2 - (\sum_{i=1}^{N} X(i))^2]\} \{ N [\sum_{i=1}^{N} X(i)^2 - (\sum_{i=1}^{N} Y(i))^2]\}}}$$

N שווה למספר התצפיות (בדוגמא זו - 10), X(I) הוא הערך של תצפית I של משתנה X ו-Y(I) הוא הערך של תצפית I של משתנה Y.

(11) משווק רוצה לעשות סיכום של המכירות במשך השנה לפי פריטים וחודשי מכירה.

כתוב תכנית שמקבלת כקלט מספר פריט, כמות שנמכרה וחודש המכירה ומפיקה דוח, כפי שניתן בדוגמא.

הנתונים ממויינים לפי מספר פריט, אבל בתוך כל קבוצת פריט אין מיון לפי חודשי המכירה ויש לקחת עובדה זו בחשבון בעת התכנות.

קלט לדוגמא מוצג להלן:

מספר פריט	כמות	חודש
100	10,000	3
100	70,000	2
100	15,000	4
100	22,000	3
427	41,000	3
427	56,000	4
427	61,000	3
427	2,000	10
427	48,000	5

הפלט ייראה כך (אפשר להציג את הנתונים לכל פריט בשורה אחת):

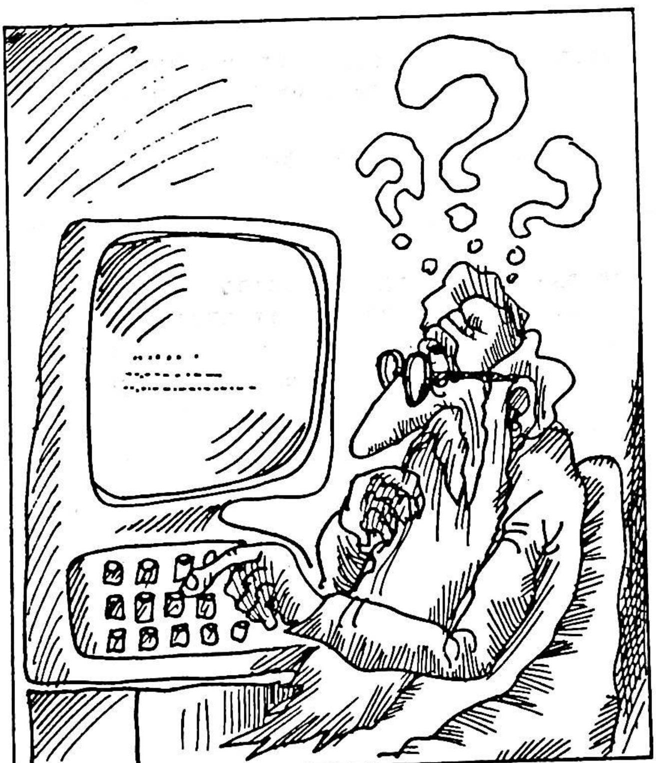
מספר פריט	ינואר	פברואר	מרס	אפריל	מאי	יוני	יולי
100	0	70,000	32,000	15,000	0	0	0
427	0	0	102,000	56,000	48,000	0	0

אוגוסט	ספטמבר	אוקטובר	נובמבר	דצמבר
0	0	0	0	0
0	0	2,000	0	0

(12) בתרגיל זה נתוני הקלט הם הציונים של 4 סטודנטים ב-5 בחינות. כתוב תכנית שתציג בטבלה את המידע הבא:

- ציון של כל סטודנט בכל מבחן,
- ציון ממוצע של כל סטודנט בכל המבחנים,
- ציון ממוצע של הסטודנטים עבור כל אחת מ-5 הבחינות,
- ציון ממוצע של כל הסטודנטים בכל הבחינות.

בתכנון הטבלה קבע את השורות לסטודנטים ואת העמודות - לבחינות.



פרק 6

נתונים לא מספריים

6.1 משתנים מסוג תו

המשתנים אשר עסקנו בהם עד עתה השתייכו לאחד משלשה סוגים של נתונים סטנדרטיים אשר מוגדרים בפסקל, או לסוגי נתונים נוספים שהוצהרו ע"י המתכנת בהצהרת TYPE או בהצהרת VAR.

שלשת הסוגים הסטנדרטיים הם INTEGER, REAL ו-BOOLEAN. סוג סטנדרטי רביעי הוא - "תו", או CHAR (מלשון Character), אשר מיועד לייצוג תווים כלשהם ולאו דווקא ערכים מספריים. משתנה מסוג תו יכול להכיל תו אחד בלבד.

תו הוא סמל כלשהו כמו אות, ספרה, סימן פיסוק או סימן מוגדר אחר, אשר ניתן להציגו בצג או להדפיסו. במחשב התו מיוצג ע"י צירוף כלשהו של סיביות, כפי שנראה בהמשך. אפשר למשל להשוות סדרות של תווים זו לזו, אך אי אפשר לעשות בהן פעולות חשבוניות, גם אם הן מורכבות מתווים שכל אחד מהם הוא ספרה. הסיבה לכך היא העובדה שצורת הייצוג של מספרים לביצוע חישובים שונה מצורת הייצוג של תווים.

6.1.1 שיטות קידוד של תווים (*)

בנספח הוסבר שמספר שלם מוחזק כמספר בינרי בשניים או בארבעה בתים ושמספר ממשי מוחזק בצורת מנטיסה ומעריך ב-4 בתים או יותר. המחשב מחזיק תווים בזיכרון בקוד של 8 סיביות. הקודים הנפוצים היום הם: EBCDIC ו-ASCII.

א. EBCDIC (Extended Binary Coded Decimal Intercange Code)

בקוד זה משתמשים ב-8 סיביות בכדי לייצג 2^8=256 תווים שונים. הייצוג של תו מחולק לשני חלקים: איזור (Zone) וספרה (Digit), אשר כל אחד מהם מורכב מ-4 סיביות.

4 סיביות ספרה	4 סיביות איזור

מכיון שאפשר לייצג כל 4 סיביות כספרה הקסדצימלית אחת, נקבע לכל תו קוד בן שתי ספרות הקסדצימליות. הטבלה הבאה מדגימה את הקוד ההקסדצימלי עבור הספרות והאותיות. גם לשאר הסימנים יש ייצוג מתאים.

תווים	ייצוג
0 עד 9	F0 עד F9
A — I	C1 — C9
J — R	D1 — D9
S — Z	E2 — E9

כדי לייצג מספר אין צורך להחזיק את 4 סיביות השמאליות ביותר (האיזור), כי הן שוות לכל הספרות. מכאן, שאפשר לשמור נתונים מספריים במבנה עשרוני ארוז (Packed Decimal) כלומר, ב-4 סיביות ולא ב-8 סיביות, אשר דרושות למבנה עשרוני מלא. לשם הדפסה או הצגה יש להפוך מבנה זה למבנה עשרוני מלא ולבצע פעולה הפוכה אשר קרויה UNPACK. שיטת EBCDIC מקובלת בעיקר במחשבים מתוצרת IBM.

ב. ASCII (American Standard Code for Information Interchange).

קוד זה פותח בחסות מכון התקנים בארה"ב בשיתוף פעולה עם כמה יצרני מחשבים. הקוד הוא בן 7 סיביות ולכן הוא יכול לייצג $2^7=128$ תווים שונים. הסיבית השמינית בכל בית מיועדת לביקורת תקינות. בהתאם לדרישות החומרה נקבע אם מספר הסיביות "1" ("דלוק"-ON) בכל בית יהיה זוגי (Even Parity) או אי-זוגי (Odd Parity). בהתאם לכך תפקיד הסיבית השמינית הוא להשלים את מספר הסיביות כנדרש. אם בעת קריאת הנתונים המחשב מגלה שמספר הסיביות במצב "1" שונה מן הדרישה, הוא מודיע על שגיאה בהעברת הנתונים.

לדוגמא, נתון תו שהסיביות שלו הן 1101110 ומספר הסיביות "1" הוא אי-זוגי. במחשב בעל ביקורת של אי-זוגיות, תהיה הסיבית השמינית "0" ובמחשב בעל ביקורת של זוגיות תהיה הספרה השמינית "1", כדי שמספר ספרות "1" בתו יהיה זוגי.

הייצוג של כל תו ב-ASCII הוא בשלושה חלקים: ספרה בינרית 0 או 1 ואחריה שתי ספרות אוקטליות, שכל אחת מהן מייצגת 3 סיביות.

ספרה	ספרה	0
אוקטלית	אוקטלית	1

הייצוג של הספרות ב-ASCII הינו בין הערכים האוקטליים 060 ו-071. הייצוג של האותיות הוא בין 101 לבין 132.

נתאר בטבלה את התווים של האלפבית האנגלי ואת תווי הספרות ולמטה מכל תו:

- הקוד שלו ב-EBCDIC (בייצוג הקסדצימלי).
- הערך הדצימלי של קוד EBCDIC.
- הקוד שלו ב-ASCII (בייצוג אוקטלי).
- הערך הדצימלי של קוד ASCII.

התו:	A	B	C	D	E	F	G	H	I	J	K	L	M
קוד EBCDIC (H)	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4
הערך הדצימלי של קוד EBCDIC	193	194	195	196	197	198	199	200	201	209	210	211	212
קוד ASCII (O)	101	102	103	104	105	106	107	110	111	112	113	114	115
הערך הדצימלי של קוד ASCII	65	66	67	68	69	70	71	72	73	74	75	76	77

התו:	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
קוד EBCDIC (H)	D5	D6	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9
הערך הדצימלי של קוד EBCDIC	213	214	215	216	217	226	227	228	229	230	231	232	233
קוד ASCII (O)	116	117	120	121	122	123	124	125	126	127	130	131	132
הערך הדצימלי של קוד ASCII	78	79	80	81	82	83	84	85	86	87	88	89	90

התו:	0	1	2	3	4	5	6	7	8	9
קוד EBCDIC (H)	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9
הערך הדצימלי של קוד EBCDIC	240	241	242	243	244	245	246	247	248	249
קוד ASCII (O)	060	061	062	063	064	065	066	067	070	071
הערך הדצימלי של קוד ASCII	48	49	50	51	52	53	54	55	56	57

H ≡ HEXA.

O ≡ OCTAL

6.1.2 הגדרת תווים בתכנית (*)

כדי להגדיר בתכנית קבוע מסוג CHAR חייבים להקיף אותו בגרשים. למשל, אם המשתנה A הוא מסוג CHAR, מותר לתת לו ערך התחלתי של T בפקודת ההצבה A:='T'. לדוגמא, קטע תכנית:

```
VAR
    A:  CHAR;
    B:  INTEGER;
BEGIN
    A:=  '7';
    B:=   7;
```

יש הבדל בין התוכן של המשתנים A ו-B לאחר פקודות הצבה אלו. ב-A נקבל בקוד את הייצוג של התו 7: F7 בשיטת EBCDIC, או 067 בשיטת ASCII. ב-B נקבל את הייצוג הבינרי של המספר 7: 00000111 או 07 בייצוג הקסדצימלי.

אפשר לייחס שני ערכים מספריים זה לזה בעזרת סמל מתאים כדי לקבל תנאי, שהוא ביטוי בוליאני. למשל, לאחר ביצוע קטע התכנית הקודם, לתנאי B<7 יהיה ערך FALSE.

גם משתנים לא מספריים יכולים להשתתף בביטוי בוליאני, אך יש לזכור שמשתמשים להשוואה בקוד הייצוגי שלהם. למשל, לתנאי A<'9' יש ערך TRUE לאחר ביצוע הקטע הקודם, בו הצבנו '7' ל-A. הייצוג של '7' ב-EBCDIC הוא F7 וערכו העשרוני הוא 247. הייצוג של '9' ב- EBCDIC הוא F9, שערכו העשרוני 249. לכן הערך '7' קטן מ-'9'.

נניח שמציבים את התו "L" במשתנה A, כלומר A := 'L'. הערך של התנאי A<'C' הוא FALSE, כי הייצוג של 'L' איננו קטן מן הייצוג של 'C', הן ב-EBCDIC והן ב-ASCII.

עתה נניח שמציבים A:='7'. מהו הערך של התנאי A<'T' ? התשובה לשאלה זו מותנית בשיטת הקידוד במחשב:

א. בשיטת EBCDIC, הייצוג של '7' הוא F7 (ערך עשרוני 247) והייצוג של 'T' הוא E3 (ערך עשרוני 227). לכן, הערך של '7' יותר גדול והתנאי A<'T' שווה ל-FALSE.
ב. בשיטת ASCII, הייצוג של '7' הוא 067 (55) והייצוג של 'T' הוא 124 (84) ולכן A<'T' שווה ל-TRUE.

אפשר להיווכח מדוגמא זו שהסדר היחסי של התווים שונה בשיטות ייצוג שונות. סדר זה נקרא סדר המיון (Collating Sequence). מן הראוי לציין שבכל סדר מיון נשמר יחס הגיוני, כלומר סדר מיון עולה בין הספרות לבין עצמן ובין האותיות לבין עצמן. השינויים העיקריים הם בהשוואת תווים מקבוצות שונות כמו למשל, ספרה עם אות, או תו מיוחד עם ספרה וכדומה. בשיטת EBCDIC הערכים של המספרים גדולים מערכי האותיות ובשיטת ASCII ערכי המספרים קטנים מערכי האותיות.

לפעמים חשוב לדעת את הערך העשרוני של תו כלשהו במחשב, או להפוך ערך מספרי לתו. קיימות שתי פונקציות שמסייעות בכך:

א. הפונקציה ORD נותנת את הערך העשרוני של התו שרשום כארגומנט של אותה פונקציה. למשל, ORD(TAV) הוא הערך העשרוני של התו שנמצא במשתנה TAV. לדוגמא, ORD('L')=76 במחשב הפועל ב-ASCII.

ב. הפונקציה ההופכית היא CHR, אשר נותנת את התו שהערך העשרוני שלו שווה לפרמטר של CHR. למשל, CHR(76) הוא התו C. הערך העשרוני של הפרמטר של CHR לא יכול לעלות על 127 במחשב ASCII, או על 255 במחשב EBCDIC.

הפונקציות CHR ו-ORD הן פונקציות הפוכות ולכן ניתן לכתוב:

א. CHR(ORD(TAV)) = TAV
ולדוגמא, CHR(ORD('L')=CHR(76) = 'L'

ב. ORD(CHR(MISPAR)) = MISPAR
ולדוגמא, ORD(CHR(76))=ORD('C') = 76

נשתמש בפונקציה CHR כדי להדגים הדפסה של כל התווים שאפשר לייצג במחשב ששיטת הייצוג שלו היא ASCII. אפשר להדפיס רק את התווים שערכם העשרוני בין 32 לבין 127, כי התווים שערכם קטן מ-32 אינם מיועדים לייצוג אלא לתקשורת בין המחשב והמסוף. הפקודות להדפסה:

```
FOR  I:=32  TO  127  DO
   WRITELN (I, '^':4, CHR(I))
```

קטע מן הפלט ייראה כך:

```
^^^^^^^^43^^^^+
^^^^^^^^44^^^^,
^^^^^^^^45^^^^-
^^^^^^^^46^^^^.
^^^^^^^^47^^^^/
^^^^^^^^48^^^^0
^^^^^^^^49^^^^1
|— 10 —|–4–|1
```

6.1.3 קריאה, כתיבה ועיבוד של תווים

נניח שנתונים שישה משפטים. ידוע שהראשון, השלישי והחמישי הם שאלות, השני, הרביעי והשישי הם תשובות. אחרי המלה האחרונה בכל משפט יש נקודה או שיש Carriage Return (במסוף, הכוונה היא למקש שכתוב עליו Return ושיוצר תו שמסמן סוף שורת קלט).

התכנית צריכה לבצע שלוש פעולות:
א. להדפיס כל שורה של קלט.
ב. להוסיף סימן שאלה לשאלות וסימן קריאה לתשובות.
ג. לספור את מספר התווים בכל שורה (בלי הנקודה בסוף המשפט, אם יש) ולהדפיס את התוצאה. רווחים לפני המשפט לא יודפסו ולא יכללו בחישוב אורך המשפט.

```
(1)  PROGRAM MISHPATIM (INPUT, OUTPUT);
(2)  CONST
(3)    NEKUDA = '.';
```

```
(4)     SIMAN_SHELA = '?';
(5)     SIMAN_KRIA = '!';
(6)     REVACH = '^';    (* הסימן ^ מיועד לציון של תו ריק בלבד *)
(7)   VAR
(8)     MONE, I : INTEGER;
(9)     TAV : CHAR;
(10)  BEGIN
(11)     FOR I:=1 TO 6 DO                    (* יש שש שורות *)
(12)       BEGIN                             (* קריאת משפט *)
(13)         MONE:=0;
(14)         WRITELN ('קרא את המשפט')
(15)         REPEAT
(16)            READ (TAV)  (* מסלקים את הרווחים המיותרים *)
(17)         UNTIL TAV <> REVACH
(18)         WHILE (TAV <> NEKUDA) AND (NOT EOLN) DO
(19)            BEGIN
(20)               MONE:= MONE+1;
(21)               WRITE (TAV);
(22)               READ (TAV)
(23)            END;
(24)         IF TAV <> NEKUDA   (* כותבים את התו האחרון *)
(25)            THEN
(26)               BEGIN
(27)                  MONE:= MONE+1;
(28)                  WRITE (TAV)
(29)               END;
(30)         IF ODD(I)  THEN
(31)             WRITELN (SIMAN_SHELA)
(32)         ELSE
(33)             WRITELN (SIMAN_KRIA);
(34)         WRITELN (MONE, 'מספר התווים בשורה:^');
(35)         READLN
(36)      END
(37)  END.
```

<u>הסבר:</u>

(3)-(6) מגדירים 4 קבועים מסוג CHAR ומציבים בהם ערכים. האורך של קבוע מסוג CHAR בהצהרת CONST, או של משתנה מסוג CHAR בפקודת השמה, חייב להיות אחד (1). תו צריך להופיע בתכנית בתוך גרשים בודדים. אם כותבים בפקודת הצבה TAV:=A כאשר A איננו מוקף בגרשים, המשמעות היא ש-A הוא שמו של משתנה אחר, אשר מתכוונים להציב את ערכו ב-TAV. כדי להציב את התו A עצמו יש לתחום אותו בגרשים.

מבחינת המחשב, רווח הוא תו ככל תו אחר וניתן להגדיר אותו בשם, כמו שעשינו בפקודה (6). אם הערך שרוצים לתת למשתנה (או לקבוע) הוא גרש, כופלים אותו בתוך הגרשים. נכתוב למשל ''''=:TAV כדי להציב ב-TAV את הערך של גרש.

(16) בכל פעם שמופעלת הפקודה READ(TAV) המחשב קורא את התו הבא בשורת הקלט במסך ולא יותר. קריאה זו שונה מקריאת מספרים שלמים או ממשיים. בקריאת מספר, המחשב מדלג על כל הרווחים בשורה עד שהוא מוצא מספר שלם או ממשי וקולט את כל התווים אשר שייכים לאותו מספר. הזיהוי של סוף המספר הוא פשוט, כי לאחר הספרה האחרונה של המספר יש תו כלשהו אשר איננו יכול להיות שייך לו, למשל אות (חוץ מ-E, אשר מייצג אקספוננט), רווח, או סימן מיוחד.

בסוג הנתונים CHAR, רווח הוא תו חוקי. לכן אין צורך להמשיך בסריקה לאחר קריאת התו הראשון, ולא חשוב מה ערכו. כאשר קוראים נתונים מן החוץ כותבים את התו במסך בלי גרשים, אע"פ שבהגדרה הפנימית בתכנית חייבים לכתוב גרשים. הם נדרשים כדי להבדיל בין תו כערך קבוע ותו כמשתנה. אבל בקריאה מן החוץ, האפשרות היחידה היא שקוראים תו כערך קבוע ולכן אין צורך בגרשיים. בסוגי הנתונים REAL ו-INTEGER אין הבדל בין הצגה פנימית בתכנית לבין הצגה חיצונית בעת הקלט, כי בכל מקרה עוסקים בערך קבוע.

(17)-(15) כדי לסלק תווי רווח מובילים, יוצרים לולאה שקוראת תווים מבלי לטפל בהם. כשהיא נתקלת בתו שונה מרווח היא מחזיקה בו לטיפול בהמשך.

(23)-(18) איך נדע מתי מסתיימת קריאת שורה? אם המשפט מסתיים בנקודה, ערכו של TAV יהיה נקודה ונוכל לבדוק זאת. אם המשפט מסתיים ב-Carriage-Return, נוכל להעזר בפונקציה בוליאנית מיוחדת, EOLN(INPUT), ראשי תיבות עבור End Of LiNe: סוף שורה. ערכה תמיד FALSE, חוץ מאשר אחרי קריאת התו שלפני Carriage Return, כאשר ערכה הופך ל-TRUE.

ברירת המחדל עבור הארגומנט של EOLN היא INPUT ולכן נהוג לא לכתוב אותו במפורש. כאשר התוכנית מבצעת READ(TAV), לאחר שקראה את התו האחרון בשורה, TAV מקבל ערך של "רווח" כאילו צורף תו רווח לשורת הקלט והפונקציה EOLN תחזור להיות FALSE. בפעם הבאה כאשר תתבצע הפקודה READ(TAV) נקבל כקלט את התו הראשון בשורה הבאה.

סימן "סוף שורה" מוגדר במסוף ע"י התו Carriage Return ולא ע"י החלוקה הפיסית לשורות. לכן, שורה לוגית אחת יכולה להשתרע על פני כמה שורות פיסיות, אם לא נכלל בה Carriage Return.

(24) עד לפקודה זו התכנית קראה את התו האחרון בשורה, אבל לא כתבה אותו. אם התו הוא נקודה אין צורך לכותבו, שהרי מחליפים אותו בסימן שאלה או בסימן קריאה. אבל אם הוא תו אחר והלולאה הופסקה כי המשפט הסתיים ב-Carriage Return, צריך לכתוב אותו וגם להוסיף 1 למונה.

(32)-(29) משתמשים בפונקציה בוליאנית שמקבלת כארגומנט מספר שלם ומחזירה את הערך הבוליאני TRUE אם המספר הוא אי-זוגי (ODD), אחרת היא מחזירה FALSE. רוצים להוסיף סימן שאלה רק במשפטים האי-זוגיים ולכן עושים זאת כאשר ODD(I) שווה ל-TRUE, כלומר במשפט אי-זוגי. כאשר ODD(I) שווה ל-FALSE נמצאים במשפט זוגי ולכן מוסיפים סימן קריאה.

(35) אם משפט מסתיים בנקודה, יש להפסיק את הקריאה גם באמצע השורה, כאשר סימן סוף שורה טרם נקרא. גם אם הלולאה הופסקה כתוצאה מהפעלה של הפונקציה EOLN, סימן סוף השורה טרם נקרא. כדי להתחיל את הקריאה מחדש

בשורה הבאה, משתמשים בפקודה READLN אשר מקדמת את הסמן לתחילת השורה הבאה. פקודה זו לא נחוצה, כי התוכנית ממילא מדלגת על הרווחים המובילים במשפט. לכן, אפילו אם הסמן היה מכוון אל אמצע השורה הקודמת, נדלג על כל הרווחים בסוף אותה שורה ונגיע לאותיות הראשונות בשורה הבאה. פקודה זו נרשמה רק "למען הסדר הטוב".

6.1.4 טיפול בתווים - בעיית הפלינדרום

<u>פלינדרום</u> (Palindrome) הוא כינוי למשפט, אשר אפשר לקרוא אותו משני הכיוונים ולקבל את אותו התוכן, בתנאי שמתייחסים רק לאותיות שלו ומתעלמים מרווחים ומתווי פיסוק. למשל, המשפט: "MADAM, I'M ADAM" הוא פלינדרום.

התוכנית הבאה קובעת אם משפט שאורכו עד 100 תווים ושמסתיים בנקודה הוא פלינדרום.

```
PROGRAM PALINDROME (INPUT, OUTPUT);
VAR
    OT: ARRAY[1..100] OF CHAR;
    I, ORECH: 0..100;
    J, K:1..100;
    TAV: CHAR;
BEGIN
    WRITELN ('הקש את המשפט לבדיקה. כתוב נקודה בסופו.');
    I:=0;
    REPEAT
       READ (TAV);
       IF (TAV >= 'A') AND (TAV <= 'Z') THEN
          BEGIN
             I:=I+1;
             OT[I]:=TAV
          END
    UNTIL TAV = '.';
    ORECH:=I;
          (* בדוק אם התוכן של המערך OT מהווה PALINDROME *)
    J:=1;
    WHILE (J<I) AND (OT[J]=OT[I]) DO
        BEGIN
           J:=J+1;
           I:=I-1
        END;
    FOR K:=1 TO ORECH DO
        WRITE (OT[K]);
    IF (J>=I)  THEN
        WRITE ('^IS')
    ELSE
        WRITE ('^ISN''T');
    WRITELN ('^A^PALINDROME')
 END.
```

הרצות לדוגמא:

הקש את המשפט לבדיקה. כתוב נקודה בסופו.
HANNAH.
HANNAH IS A PALINDROME

הקש את המשפט לבדיקה. כתוב נקודה בסופו.
ABCDEFGH.
ABCDEFGH ISN'T A PALINDROME

הסבר:

(10)-(17) קריאת המשפט לתוך המערך OT. מכניסים למערך אותיות בלבד.

(18) מספר התווים שמכניסים למערך מאוחסן ב-ORECH.

(20)-(25) משווים את האות הראשונה אל האות האחרונה. אם אין שיוויון, מפסיקים את ההשוואה. אם הן שוות, משווים את האות השנייה אל השנייה מן הסוף וכן הלאה. ההשוואה תפסק בגלל אחת מן הסיבות הבאות:

א. כל אות מתחילת המשפט שווה לכל אות במקום מקביל מסוף המשפט. הלולאה מפסיקה כאשר מסיימים את ההשוואה של החצי הראשון של המשפט אל החצי השני.
ב. יש אי-שיוויון בין אחת מן האותיות מתחילת המשפט לבין האות המקבילה מסוף המשפט, ז"א פגשנו בזוג של אותיות אשר שונות זו מזו.

(28)-(31) אם היציאה מן הלולאה היא בגלל סיבה (א), אז המשפט הוא פלינדרום ונכתוב IS. אם יוצאים מן הלולאה בגלל הסיבה (ב), אז המשפט איננו פלינדרום ונכתוב ISN'T. היות והמלה ISN'T נמצאת בתוך גרשים, צריך לכתוב את הגרש של ISN'T פעמיים (כלומר ISN''T), כדי שהוא לא יתקבל כסוף המחרוזת.

6.2 הפקודות READ ו-WRITE

קריאת תווים מן המסוף נעשית תו אחר תו, פרט לקלט אשר מוגדר כמספר שלם או ממשי. במקרה זה הפקודה READ יודעת להמשיך ולקרוא תווים (שהם ספרות המספר) עד שהיא נתקלת ברווח או פסיק ומפענחת תווים אלה כמספר בן ספרות אחדות.

מן הראוי להדגיש שהכוונה כאן היא לקריאת נתונים מתוך מאגר הקלט שבזיכרון אל המשתנים של התוכנית. תהליך הקריאה הפיסי נעשה בדרך שונה במסופים מסוגים שונים. יש מסופים הפועלים בשיטת Start-Stop ומזינים את הזיכרון בתווים בודדים בכל הקשה. לעומתם יש מסופים הפועלים בשיטת Full Screen, אשר מזינים לזיכרון את כל אשר הוקש מאז ה-Carriage Return, או Enter.

כדי לדמות את הפקודה READ בקריאת מספרים שלמים, מוצגת תכנית שקוראת מספר כסדרה של תווים ומתרגמת אותו למספר עשרוני המורכב מאותה סדרה של תווים.

```
(1)     PROGRAM KRIA (INPUT, OUTPUT);
(2)     VAR
(3)        TAV: CHAR;
(4)        MISPAR: INTEGER;
(5)     BEGIN
(6)        MISPAR:=0;
(7)        READ (TAV);
(8)        WHILE TAV<> '^' DO
(9)          BEGIN
(10)           MISPAR:= 10*MISPAR + ORD(TAV) - ORD('0');
(11)           READ (TAV)
(12)         END;
(13)       WRITELN (MISPAR)
(14)    END.
```

<u>הסבר:</u>

(10) מוצאים את הערך העשרוני של התו שנקרא. למשל, אם התו הוא הספרה 5, ערכו העשרוני בשיטת ASCII הוא 53. מפחיתים ממנו את ORD('0'), שערכו העשרוני הוא 48. הערך החדש של MISPAR הוא 10*0+53-48=5. אם התו שקוראים אחריו הוא 9, שערכו העשרוני הוא 57 ב-ASCII, הערך החדש של MISPAR יהיה: 10*5+57-48=59. תהליך זה נמשך עד שמגיעים לתו רווח.

(13) לבדיקת ההצלחה של התהליך מדפיסים את המשתנה MISPAR כמספר שלם. לעתים קרובות מוסיפים פסיק בתוך תווי המספר, למשל 5,817. הפקודה READ אינה יכולה לקרוא מספר שיש בו סימן הפרדה כזה ותקבל אותו כשני מספרים: 5 ו-817. אפשר לערוך מספר תיקונים בתכנית זו כדי שתוכל לקלוט מספר המיוצג עם סימני הפרדה בתוכו. לא נעסוק בכך כאן אלא באחד מהתרגילים שבסוף הפרק.

בזמן כתיבה מתורגם כל מספר מצורת ייצוג בינרית לצורת ייצוג בתווים. הפלט נשלח למסוף כתווים בודדים בקוד המתאים לסוג המסוף, כאשר פועלים במסוף Start-Stop. במסוף מסוג Full-Screen, הפלט נשלח בשדות, בשורות או במסך שלם בהתאם לעניין.

התכנית הבאה מדמה את פעולת הפקודה WRITE:

```
PROGRAM KTIVA (INPUT, OUTPUT);
VAR
  MISPAR I, J: INTEGER;
  MARACH: ARRAY[1..10] OF CHAR;
BEGIN
  WRITELN ('הקש מספר');
  READLN (MISPAR);
  I:=1;
  WHILE (MISPAR<>0) AND (I<=10) DO
     BEGIN
        MARACH[I]:= CHR (ORD('0')+MISPAR MOD 10);
        MISPAR:= MISPAR DIV 10;
        I:=I+1
     END;
```

```
(15)    J:=I-1;
(16)    FOR I:=J DOWNTO 1 DO
(17)       WRITE (MARACH[I])
(18)  END.
```

<u>הסבר:</u>

(7) המספר שנקרא בפקודה READ מוחזק כערך בינרי.

(11) מוסיפים לספרה הימנית ביותר (MISPAR MOD 10) את המספר 48 שהוא הערך של '0' ב-ASCII. למשל, אם הספרה הימנית במספר היא 5 נקבל CHR(53), כאשר 53 הוא הערך העשרוני של קוד ASCII עבור התו '5'.

(12) מורידים את הספרה הימנית ביותר מן המספר ומוסיפים 1 לאינדקס I בשורה (13).

(13)-(11) ממשיכים לבצע פקודות אלו עד שמסיימים את הטיפול בכל הספרות שמרכיבות את המספר הנתון (ז"א עד ש: MISPAR=0).

(16) חייבים להתחיל להדפיס מן הספרה המשמעותית ביותר ולכן מתחילים ממקום I-1 במערך ויורדים עד למקום 1.

הפלט עבור מספר שלם הוא סדרה של ספרות. לעתים רוצים להוסיף פסיק בכדי לציין חלוקת הספרות לשלשות ולכתוב למשל 4,729 במקום 4729. לשם כך צריך לוותר על השימוש בפקודה WRITE עבור מספר ולהשתמש בתכנית הדומה לתכנית KTIVA שהצגנו למעלה. במקום המתאים מדפיסים פסיק.

<u>6.3 מחרוזות</u>

סוג הנתונים CHAR מיועד לטיפול בתווים נפרדים, אך רוב המידע האלפביתי ערוך במבנה מלים, או משפטים. השם הכללי לקבוצה של תווים כלשהם הוא <u>מחרוזת</u> (String). אפשר להגדיר מחרוזת כמערך של תווים כדי שאפשר יהיה לפנות אל כולה בשם אחד - השם של המערך. למשל,

```
TYPE
 STRING = ARRAY [1..25] OF CHAR;
VAR
  NAME1, NAME2: STRING;
```

לאחר ההגדרה, אפשר לבצע פעולות שונות על מערכים אלו.

<u>6.3.1 פעולות במחרוזות תווים</u>

א. קריאה של נתונים אל מחרוזת NAME1:

```
FOR I:=1 TO 25 DO
   READ (NAME1[I]);
READLN;
```

ב. העברה של תווים ממחרוזת NAME1 אל מחרוזת NAME2:

```
FOR I:=1 TO 25 DO
   NAME2[I]:= NAME1[I];
```

ג. הדפסה של מחרוזת NAME1:

```
FOR I:=1 TO 25 DO
   WRITE (NAME1[I]);
WRITELN;
```

ד. השוואה של מחרוזת NAME1 אל מחרוזת NAME2:
תו נחשב לקטן או גדול בערכו מתו אחר לפי מקומו <u>בסדר המיון</u> במחשב. בהשוואה של מחרוזות, המחרוזת הראשונה נחשבת כקטנה יותר בערכה מן השנייה, אם התו הראשון שלה קטן מן התו הראשון של חברתה. לעומת זאת, אם התו הראשון גדול יותר במחרוזת הראשונה, היא נחשבת גדולה מחברתה. אם התו הראשון שווה בשתי המחרוזות, משווים את התו השני ואם גם הוא שווה, ממשיכים לבדוק עד שמגיעים להכרעה. אם מסיימים את ההשוואה בלי להגיע להכרעה, המחרוזות נחשבות שוות. לכן

ניתן להשוות את NAME1 עם NAME2 ע"י הפקודות הבאות:

```
I:=1;
KATAN:= FALSE;
GADOL:= FALSE;
REPEAT
  IF NAME1[I] < NAME2[I] THEN
     KATAN:= TRUE
  ELSE
     IF NAME1[I]>NAME2[I] THEN
        GADOL:= TRUE;
  I:= I+1
UNTIL (KATAN OR GADOL) OR I>25;
IF KATAN THEN
   WRITELN ('NAME2-קטן מ NAME1')
ELSE
   IF GADOL THEN
      WRITELN ('NAME2-גדול מ NAME1')
   ELSE
      WRITELN ('NAME2-שווה ל NAME1');
```

ה. פעולות במחרוזות שלמות:
יותר נוח לעבוד עם המחרוזת NAME1 ו-NAME2 ביחידות שלמות ולא כמערכים. לשם כך, יש לכתוב במקום א' עד ד' שהופיעו למעלה את הפקודות הבאות:

```
*(1)  READLN (NAME1);
*(2)  NAME2:= NAME1;
*(3)  WRITELN (NAME1);
*(4)  IF NAME1 < NAME2 THEN
         WRITELN ('NAME2-קטן מ NAME1')
      ELSE
         IF NAME1 > NAME2 THEN
            WRITELN ('NAME2-גדול מ NAME1')
```

```
ELSE
   WRITELN ('NAME2-ל שווה NAME1')
```

רוב המהדרים מאפשרים לכתוב את הפקודות (2), (3) ו-(4) ויש שמאפשרים לכתוב גם את הפקודה (1), כאשר מגדירים את NAME1 ו-NAME2 כ-PACKED ARRAY. המלה PACKED מתייחסת לעובדה שהתווים במערך זה הם ארוזים, כדי שאפשר יהיה להחזיק אותם בצורה יותר דחוסה. כדי לגרום את אריזת הנתונים, על המתכנת רק להוסיף את המלה PACKED לפני המלה ARRAY בהצהרת TYPE או VAR. יש מהדרים שמעניקים תכונות אלו למערך גם מבלי שיוגדר כ-PACKED.

6.3.2 שימוש במחרוזות

נדגים בתכנית הבאה את השימוש במחרוזות:
התכנית מיועדת לסייע בעריכת שידוכים. נתונים פרטים אודות האנשים והנשים אשר רשומים כבר במחשב ומשתתפים בתכנית ואודות מועמדים חדשים, אשר מחפשים בני זוג מתאימים. נתוני הקלט לכל משתתף הם:

א. שם,
ב. מין,
ג. מידת האהדה לנושאים מסוימים, שנמדדת בסולם מ-1 עד 5.

בפרט ג, אם התשובה לנושא ספורט היא למשל 1, משמעותה שהמועמד/ת לא סובל/ת ספורט. אם התשובה היא 5 משמעותה להיטות אחר ספורט. תשובה 3 מציינת עמדה ניטרלית, 4 - גישה חיובית לספורט ו-2 - גישה שלילית.

בתהליך העיבוד מקבלים את הנתונים עבור מועמד/ת לחיפוש בן/בת זוג. סורקים את הנתונים של כל המשתתפים מן המין השני וקובעים איזה מהם הוא המתאים ביותר למועמד.

כדי למדוד את מידת ההתאמה בין שני אנשים נשתמש בשיטת הריבועים המינימליים. בשיטה זו מוצאים את ההפרש בין כל אחת מהתשובות של המועמד בכל נושא לבין התשובות של בן הזוג הפוטנציאלי באותם נושאים. מחשבים את ריבועי ההפרשים לכל התשובות ומחברים אותם. ככל שסכום הריבועים יותר קטן, ההתאמה נחשבת יותר טובה.

לדוגמא, התשובות של אברהם בנושאים ספורט, מוסיקה ודת היו 3, 4 ו-5 בהתאמה, התשובות של רבקה היו 3, 2 ו-2 והתשובות של לאה היו 4, 2 ו-5. על פי נתונים אלה נקבל את התוצאות הבאות:
סכום הריבועים בעבור השידוך אברהם-רבקה יהיה: $(5-2)^2+(4-2)^2+(3-3)^2=13$.
בעבור השידוך אברהם-לאה סכום הריבועים יהיה: $(5-5)^2+(4-2)^2+(3-4)^2=5$.
סכום הריבועים קטן יותר בעבור השידוך אברהם-לאה מאשר בעבור השידוך אברהם-רבקה ולכן השידוך אברהם-לאה יותר מתאים.

```
(1)  PROGRAM SHIDUCHIM (INPUT, OUTPUT);
(2)  CONST
(3)       MAX_MISHTATFIM=20;
(4)       MISPAR_SHEELOT=10;
(5)  TYPE
(6)       SHEM= PACKED ARRAY[1..20] OF CHAR;
```

```
        TSHUVA= 1..5;
        MISHTATFIM= 1..MAX_MISHTATFIM;
        SHEELOT= 1..MISPAR_SHEELOT;
    VAR
                                                  (* משתני עבודה *)
        MIN_RIBUIM: INTEGER;
        J: SHEELOT;
        I, MATIMIM, MISPAR_MISHTATFIM: MISHTATFIM;
                                                  (* פרטים אודות המועמד *)
        MIN_MUAMAD: CHAR;
        SHEM_MUAMAD: SHEM;
        TSHUVOT_MUAMAD: ARRAY[SHEELOT] OF TSHUVA;
                                                  (* מידע אודות המשתתפים *)
        MIN_MISHTATEF: ARRAY[MISHTATFIM] OF CHAR;
        SCHUM_RIBUIM: ARRAY[MISHTATFIM] OF INTEGER;
        TSHUVOT_MISHTATEF: ARRAY [MISHTATFIM,SHEELOT] OF TSHUVOT;
        SHEM_MISHTATEF: ARRAY[MISHTATFIM] OF SHEM;
        MISHTATFIM_TOVIM: ARRAY[MISHTATFIM] OF INTEGER;
    BEGIN
      WRITELN ('כמה אנשים משתתפים?');
      READLN (MISPAR_MISHTATFIM);
                                  (* הכנס פרטים אודות כל אחד מן המשתתפים *)
      FOR I:=1 TO MISPAR_MISHTATFIM  DO
        BEGIN
          WRITELN ('כתוב את שם המשתתף');
          READLN (SHEM_MISHTATEF[I]);
          WRITELN ('מין?');
          READLN (MIN_MISHTATEF[I]);
          WRITELN ('תשובות לנושאים?');
          FOR J:=1 TO MISPAR_SHEELOT DO
             READ (TSHUVOT_MISHTATEF[I,J]);
          READLN
        END;
                                                  (* טפל בכל מועמד *)
      WHILE NOT EOF DO
        BEGIN
          WRITELN ('כתוב את שם המועמד');
          READLN (SHEM_MUAMAD);
          WRITELN ('מין?');
          READLN (MIN_MUAMAD);
          WRITELN ('תשובות לנושאים?');
          FOR J:=1 TO MISPAR_SHEELOT DO
             READ (TSHUVOT_MUAMAD[J]);
          READLN;
           (* חשב את ההתאמה של המועמד לכל משתתף מן המין השני *)
          FOR I=1 TO MISPAR_MISHTATFIM DO
             SCHUM_RIBUIM[I]:=0;
          FOR I:= 1 TO MISPAR_MISHTATFIM DO
             IF MIN_MUAMAD <> MIN_MISHTATEF[I]  THEN
               FOR J:= 1 TO MISPAR_SHEELOT DO
```

```
(57)                SCHUM_RIBUIM[I]:= SCHUM_RIBUIM[I] +
(58)                 SQR (TSHUVOT_MISHTATEF[I,J] - TSHUVOT_MUAMAD[J]);
(59)                            (* מצא את סכום הריבועים המינימלי *)
(60)          MIN_RIBUIM:= MAXINT;
(61)          FOR I:= 1 TO MISPAR_MISHTATFIM DO
(62)            IF MIN_MUAMAD <> MIN_MISHTATEF[I] THEN
(63)              IF SCHUM_RIBUIM[I] < MIN_RIBUIM THEN
(64)                MIN_RIBUIM:= SCHUM_RIBUIM[I];
(65)                   (* בחר את המתאימים ביותר עבור המועמד הזה *)
(66)          MATIMIM:=0;
(67)          FOR I:= 1 TO MISPAR_MISHTATFIM DO
(68)             IF MIN_MUAMAD <> MIN_MISHTATEF[I] THEN
(69)                IF SCHUM_RIBUIM[I] = MIN_RIBUIM THEN
(70)                   BEGIN
(71)                      MATIMIM:= MATIMIM+1;
(72)                      MISHTATFIM_TOVIM[MATIMIM]:=I
(73)                   END;
(74)                             (* הוצא כפלט את המתאימים ביותר *)
(75)          WRITELN (SHEM_MUAMAD, 'שם המועמד הוא:');
(76)          WRITELN ('המשתתפים המתאימים ביותר:');
(77)          FOR I:= 1 TO MATIMIM DO
(78)            WRITELN (SHEM_MISHTATEF [MISHTATFIM_TOVIM[I]])
(79)        END
(80)  END.
```

<u>הסבר:</u>

(26)-(27) קלט של מספר האנשים שישתתפו בקבוצה הבסיסית.

(29)-(39) קלט נתונים בעבור כל אחד מן המשתתפים. מניחים שהמהדר שמשתמשים בו מאפשר לקרוא PACKED ARRAY בלי לולאה. לכן כתבנו בשורה (32): READLN (SHEM MISHTATEF[I]).

(41)-(79) לולאה גדולה שמטפלת בכל מועמד שרוצים לשדך.

(43)-(50) קריאה של נתונים אודות המועמד.

(52)-(53) איפוס של המערך SCHUM_RIBUIM.

(54)-(58) אם המין של משתתף I שונה ממין המועמד, מחשבים את סכום ריבועי ההפרשים שבין התשובות שלו לבין התשובות של המועמד. כותבים את הסכום הזה במקום I בווקטור SCHUM_RIBUIM.

(60)-(64) סורקים את הווקטור SCHUM_RIBUIM כדי למצוא את הערך המינימלי שבין ערכי כל האיברים בווקטור זה. הסכום עבור משתתפים שמינם שווה למינו של המועמד לא רלוונטי ולכן מחפשים את הסכום המינימלי אשר קשור לאנשים מן המין האחר.

(66)-(73) אילו היינו מסתפקים במשתתף אחד עבור המועמד, יכולנו להחזיק את מספרו של המשתתף שסכום הריבועים שלו הוא המינימלי. מכיון שרוצים

להתייחס לאפשרות שיש יותר ממשתתף אחד עם אותו סכום ריבועים מינימלי, סורקים פעם נוספת את הווקטור SCHUM_RIBUIM. עתה מכניסים את מספרו של כל משתתף שסכום ריבועיו שווה לסכום הריבועים המינימלי לווקטור חדש שנקרא MISHTATFIM_TOVIM.

(77)-(78) מדפיסים את השמות של המשתתפים שמספרם נמצא בווקטור בשם MISHTATFIM_TOVIM, שהם בעלי אותו סכום ריבועים מינימלי.

6.4 תרגילים

6.4.1 שאלות

(1) מה מדפיסה התכנית הבאה ?

```
PROGRAM IMRA (INPUT, OUTPUT);
TYPE
    MACHROZET = PACKED ARRAY[1..6] OF CHAR;
VAR
    SHIR: ARRAY[1..12] OF MACHROZET;
    MONE1, MONE2: INTEGER;
BEGIN
   SHIR[1]:= 'A ROSE';
   SHIR[2]:= '  IS  ';
   FOR MONE1:= 1 TO 3 DO
     FOR MONE2:= 1 TO 2 DO
         WRITELN (SHIR[MONE2]);
   WRITELN (SHIR[1])
END.
```

(2) מה מדפיסה התכנית הבאה על פי הקלט אשר היא קוראת?

```
PROGRAM TARGIL (INPUT, OUTPUT);
TYPE
      SUG_SHEM= PACKED ARRAY[1..10] OF CHAR;
VAR
      SHEM: ARRAY[1..25] OF SUG_SHEM;
      MCHIR: ARRAY[1..25] OF INTEGER;
      I, J, M, N: INTEGER;
BEGIN
      READLN (N);
      FOR I:= 1 TO N DO
         READLN (SHEM[I], MCHIR[I]);
      READLN (M);
      FOR I:= 1 TO M DO
         IF MCHIR[I] > M THEN
            WRITELN (SHEM[I])
END.
```

הקלט הוא:

```
5
SUCAR     442
MITZ      618
CHALAV    391
LECHEM    201
ESHEL     246
500
```

(3) קרא את התכנית שבסעיף 6.1.2 שבפרק זה, שבו מציגים תווים ואת ערכם העשרוני בשיטת ASCII. הוסף לה פקודות כדי להדפיס ליד הערך העשרוני גם את הערך בקוד אוקטלי.

הנחיה: הפוך מספר עשרוני למספר אוקטלי ע"י חלוקה נשנית ב-8 והחזקת השארית כספרה אוקטלית עד שהמספר המקורי יורד ל-0. למשל, אם המספר העשרוני הוא 43 עשה את הצעדים הבאים:

א. 43/8 שווה ל-5 עם שארית 3 (3 היא הספרה הימנית בייצוג אוקטלי).
ב. 5/8 שווה ל-0 עם שארית 5 (5 היא הספרה השנייה מימין).
ג. התוצאה היא היצוג האוקטלי 53_8 של המספר העשרוני 43.

(4) קרא את התכנית KRIA שבסעיף 6.2 בפרק זה שקוראת מספר כסדרת תווים. הוסף לה פקודות כדי שיהיה אפשר לקרוא כקלט מספר שלם עם פסיק לסימון אלפים.

(5) קרא את התכנית KRIA שבסעיף 6.2 והוסף לה פקודות כדי שהיא תכלול פסיק לסימון אלפים בזמן הדפסה של מספר שלם (אם המספר גדול מ-999).

6.4.2 תכניות

(1) כתוב תכנית שקוראת תמליל באנגלית ומדפיסה אותו כפלט, כך שכל אות מוחלפת ע"י האות שבאה אחריה באלפבית: האות C מוחלפת ע"י האות D וכו' ו-Z מוחלפת ע"י A.

(2) כתוב תכנית שקוראת רשימה של שמות ומדפיסה אותם בסדר הפוך. האורך המקסימלי של שם הוא 25 תווים.

(3) כתוב תכנית שקוראת שם של אדם, את גובהו בס"מ ואת משקלו בקילוגרמים. היא מדפיסה את הנתונים שקיבלה ובנוסף מחשבת ומדפיסה את ממוצע מספר הקילוגרמים לכל מטר גובה.

(4) הקלט עבור תכנית שירות לבורסה מכיל 3 נתונים:

	(א) שם המניה	(ב) מחיר היום	(ג) מחיר אתמול
למשל,	פיתוח לייזר	139	137
	תעשיית מחשבים	2431	2387

כתוב תכנית שמדפיסה רשימה של המניות שעלו או ירדו ביותר מ-10%.

(5) הכלל בכתיב אנגלי הוא שהאות I קודמת לאות E, חוץ מן המקרה שזוג זה בא אחרי האות C ואז הסדר צריך להיות E לפני I. כתוב תכנית שקוראת תמליל ובודקת אם כלל זה מתקיים. אם הוא לא מתקיים התכנית מדפיסה את המלה השגויה.

(6) העלות למשלוח חבילות דואר בארץ מבוססת על 3 קריטריונים:

א. מקום היעד - M - מקומי.
E - עיר גדולה (ירושלים, ת"א, חיפה, באר-שבע).
K - עיר קטנה, מושב, קיבוץ וכו'.
ב. משקל - K - קל (פחות מקילו).
M - ממוצע (בין קילו ל-10 קילו).
C - כבד (למעלה מ-10 קילו).
ג. שירות - R - רגיל.
E - אקספרס.

התעריף למשלוח חבילה נמדד לפי נקודות, אשר נקבע להן מידי פעם ערך שקלי כדי לדעת את התשלום בפועל. הטבלה שלמטה מאפשרת לחשב את המחיר למשלוח חבילה: בחלק העליון של הטבלה מופיעות כל התמורות השונות של יעד, משקל וסוג שירות. בחלק התחתון של הטבלה נרשם התעריף בנקודות.

יעד:	M	M	M	M	M	M	E	E	E	E	E	E	K	K	K	K	K	K
משקל:	K	K	M	M	C	C	K	K	M	M	C	C	K	K	M	M	C	C
שירות:	E	R	E	R	E	R	E	R	E	R	E	R	E	R	E	R	E	R
6 נקודות		X						X										
12 נקודות	X						X						X	X				
50 נקודות +2 לקילו מעל ל-20						X												X
100 נקודות +2 לקילו מעל ל-20					X												X	
2 נק'/קילו				X														
3 נק'/קילו			X							X		X						
6 נק'/קילו									X		X					X		
8 נק'/קילו															X			

כתוב תכנית שמקבלת כקלט את מקום היעד (M, E, K), משקל בקילו, סוג השירות (E,R) והערך השקלי של "נקודה". התכנית מפיקה כפלט את המחיר למשלוח החבילה.

(7) כתוב תכנית שמקבלת כקלט שורות של תמליל באנגלית, מכניסה אותן למערך ומדפיסה אותן. אם צירוף התווים "N#^" מופיע בתחילת השורה, היא תודפס בהזזה של N מקומות לימין, כאשר N הוא מספר בין 1 לבין 120.

(8) כתוב תכנית שקוראת זוגות של מלים באנגלית ומדפיסה אותן אחת ליד חברתה לפי סדר אלפבתי, משמאל לימין. למשל, שתי המלים Very Good תודפסנה בסדר הפוך, אבל שתי המלים Big Dog תשארנה בסדר זה. דאג לכך שבפלט יהיו 5 רווחים בין המלה הראשונה והשנייה ושהזוגות יהיו ממוספרים החל מ-1.

(9) קרא משפט, אשר מסתיים בנקודה, ותכנת את הפעולות הבאות:
א. מצא את מקומו של הרווח הראשון והכנס אותו למשתנה בשם REVACH_RISHON.
ב. מצא את האות הראשונה במלה האחרונה והכנס אותה למשתנה בשם MILA_ACHRONA.
ג. ספור את המלים במשפט.

(10) התוכנית הבאה מטפלת באיתור מבצעי פשע על פי שתי קבוצות של נתונים:
א. כרטסת הפושעים הידועים למשטרה, שכוללת את הנתונים: שם, גובה, משקל.
ב. פשעים שהתבצעו לאחרונה ומתוארים ע"י הנתונים הבאים: תיאור הפשע, גובה ומשקל משוערים של הפושע.

כתוב תכנית שתדפיס את תיאור הפשע ואת שמות החשודים בביצועו. החשודים הם אנשים שגובהם חורג בפחות מ-5 ס"מ מהגובה המשוער ומשקלם חורג בפחות מ-5 קילו מהמשקל המשוער של מבצע הפשע.

(11) (*) כתוב תכנית אשר סורקת תמליל באנגלית בכדי למצוא את השכיחות היחסית של המופעים של כל אות.
בנה מערך: SHCHICHUT[1..26] OF INTEGER
הוסף 1 לאיבר SHCHICHUT[I], כאשר ORD(TAV)-ORD('A')+1 שווה ל-I בעבור כל תו (TAV) אשר נקרא כקלט.
קח בחשבון שהתכנית צריכה לחבר יחד מופעים של אות מסוימת כאות גדולה וגם כאות קטנה.

(12) כתוב תכנית אשר קוראת סדרה של 10 מספרים רצופים בני שלושה תווים כל אחד ומחשבת ומדפיסה את הסכום שלהם. שלושת התווים של כל מספר יכולים להכיל ספרות, סימן + או – ורווחים, באחד מן הצירופים הבאים:
- 3 ספרות.
- רווח או סימן ושתי ספרות.
- רווח, רווח או סימן, ספרה.

קלט לדוגמא נראה כך:

```
+11^47732-71^^4971-19402^81414
1  2  3  4  5  6  7  8  9  10
```

(13) כתוב תכנית שקוראת ומדפיסה תמליל ומחשבת את הערכים הבאים:
- מספר השורות בתמליל.
- מספר המשפטים.
- מספר המלים.

אפשר להניח שכל משפט מסתיים בנקודה-רווח וזה השימוש היחיד של זוג תווים זה. אפשר גם להניח שמלה מורכבת אך ורק מאותיות.

(14) (*) בשורת הנתונים הראשונה מופיע המספר N ולאחריו N מחרוזות בנות 10 תווים כל אחת. כתוב תכנית אשר:
- קוראת את N המחרוזות.
- מחשבת את מספר הפעמים שהתו T מופיע בכל אחת מהמחרוזות.
- מדפיסה את המספר הסידורי של המחרוזת המכילה את המספר המירבי של מופעי התו T. אם יש שיוויון בין שתי מחרוזות, אפשר להדפיס את המספר הסידורי של אחת מהן בלבד.
- מדפיסה בכמה מחרוזות ההופעה הראשונה של האות S היא לפני האות T, בתנאי ששתי האותיות מופיעות.

(15) (*) כתוב תכנית שקוראת שורות של מכתב לתוך מערך. בראש ובגוף המכתב מופיעים הסימונים *NAME* ו-*DATE*. בסוף המכתב, ישנן שורות קלט נוספות שבכל אחת מהן כתוב זוג של מלים, שהראשונה היא שם והשנייה - תאריך. הפלט צריך להיות מכתבים מוכנים, כאשר בכל מכתב ירשמו השם והתאריך שהתקבלו בקלט במקום המלים *NAME* ו-*DATE* בהתאמה. מספר המכתבים המהווים את הפלט הוא כמספר הזוגות של שם ותאריך בקלט.

(16) (*) כתוב תכנית שמקבלת כקלט קטע תמליל ומדפיסה אותו בשורות עד 60 תווים. הסימן "/" מציין התחלת פיסקה חדשה, שבדוגמא זו תתחיל מן המקום הרביעי בשורה חדשה. התו "/" לא יודפס בפלט. אסור לחלק מלה לשניים. סימני פיסוק כמו נקודה או פסיק נחשבים חלק מן המלה הקודמת.

בפתרון הבעיה יש לטפל בשלושה מקרים שונים:
- מדפיסים שורה מלאה,
- מגיעים לסימן "/",
- מגיעים לסוף הקטע.



פרק 7

חיפוש, מיון ומיזוג

7.1 מבוא

בפרק 6 למדנו להחסן נתונים במערך חד-מימדי, או ווקטור. בפרק זה נלמד לבצע שלוש פעולות נפוצות על האיברים של מערך: חיפוש, מיון ומיזוג. מעריכים שכמחצית מזמן העבודה של מחשבים מוקדש לפעולות אלו.

א. חיפוש (Searching): סריקה של כל האיברים במערך כדי למצוא את האיבר, או האיברים, שמקיימים תנאי מסוים, או שהם שווים לערך מסוים.

ב. מיון (Sorting): סידור האיברים במערך לפי הערך שלהם, בסדר עולה מקטן לגדול, או בסדר יורד מגדול לקטן.

ג. מיזוג (Merging): איחוד של שני מערכים ממויינים לתוך מערך חדש. מערך זה גם ממויין וגם מכיל את כל האיברים שהיו בשני המערכים המקוריים.

האיברים במערכים יכולים להיות מספרים שלמים, ממשיים, תווים או שמות (ז"א מערכים של תווים).

7.2 חיפוש

7.2.1 חיפוש ליניארי

יש שתי שיטות נפוצות לחיפוש איברים במערך. השיטה הפשוטה ביותר היא חיפוש ליניארי (Linear Search). חיפוש זה מתחיל באיבר הראשון במערך ומתקדם לקראת סופו, עד שנתקלים באיבר הרצוי.

נדגים בתכנית את שיטת החיפוש הליניארי:
מחפשים את המספר של חבר ברשימת מספרי טלפון, אשר מסודרת בסדר אלפביתי.

```
(1)  PROGRAM MISPAREI_TELEPHONE (INPUT, OUTPUT);
```

```
(2)  TYPE
(3)     SHEM= PACKED ARRAY[1..20] OF CHAR;
(4)     MISPAR= PACKED ARRAY[1..10] OF CHAR;
(5)  VAR
(6)     I, GODEL_RESHIMA: 1..100;
(7)     RESHIMAT_MISPARIM: ARRAY[1..100] OF MISPAR;
(8)     RESHIMAT_SHEMOT: ARRAY[1..100] OF SHEM;
(9)     SHEM_CHAVER: SHEM;
(10)    NIMTZA: BOOLEAN;
(11) BEGIN
(12)    READ (GODEL_RESHIMA);
(13)                          (* קרא את השמות והמספרים שברשימה *)
(14)    FOR I:= 1 TO GODEL_RESHIMA DO
(15)       BEGIN
(16)         READLN (RESHIMAT_SHEMOT[I]);
(17)         READLN (RESHIMAT_MISPARIM[I])
(18)       END;
(19)    WHILE NOT EOF DO
(20)       BEGIN
(21)         READLN (SHEM_CHAVER);
(22)         NIMTZA:= FALSE;
(23)         I:=1;
(24)         WHILE (I <= GODEL_RESHIMA) AND NOT NIMTZA DO
(25)            IF (SHEM_CHAVER = RESHIMAT_SHEMOT[I]) THEN
(26)                NIMTZA:= TRUE
(27)            ELSE
(28)                I:= I+1;
(29)         WRITELN (SHEM_CHAVER, '^':10);
(30)         IF NIMTZA THEN
(31)            WRITELN (RESHIMAT_MISPARIM[I])
(32)         ELSE
(33)            WRITELN ('לא נמצא ברשימה')
(34)       END
(35) END.
```

<u>הסבר:</u>

(12)-(18) קריאה של רשימת השמות והמספרים. מניחים שהמהדר מאפשר לקרוא "PACKED ARRAY OF CHAR" בלי לולאה.

(19)-(34) חיפוש מספר של חבר ברשימת המספרים (אם יש).

(24)-(28) לכאורה, היה יותר קל להחליף שורות אלו בפקודות יותר פשוטות:

```
WHILE (I <= GODEL_RESHIMA) AND
                    RESHIMAT_SHEMOT[I] <> SHEM_CHAVER) DO
    I:= I+1;
```

בפקודות אלו מתחילים את החיפוש באיבר RESHIMAT_SHEMOT[1] ומשווים אותו ל-SHEM_CHAVER. אם השמות לא שווים, ממשיכים לסרוק את הרשימה עד שאחד משני מאורעות מתרחש:

א. SHEM_CHAVER שווה ל-RESHIMAT_SHEMOT[I].

ב. מסיימים לקרוא את הרשימה.

היתרון של שיטה זו הוא בכך שאין צורך במשתנה הבוליאני NIMTZA. החיסרון שלה הוא בכך שהיא עלולה לגרום לשגיאת ביצוע במהדרים מסוימים, אם השם שמחפשים איננו נמצא ברשימה. במקרה זה, I יגיע לערך GODEL_RESHIMA+1, למשל 101 בדוגמא זו, אשר נמצא <u>מחוץ</u> לתחום המערך.

כאשר זה קורה, התנאי:
(I <= GODEL_RESHIMA) AND (RESHIMAT_SHEMOT[I] <> SHEM_CHAVER)
יהיה FALSE, מכיון שהחלק הראשון של הביטוי I<=GODEL_RESHIMA איננו קיים. החלק השני של התנאי יהיה אז RESHIMAT SHEMOT[101]. אינדקס אשר נמצא מחוץ לתחום הנכון (למשל 101) יגרום להודעת שגיאה. השאלה היא, אם המהדר יבדוק את החלק השני של התנאי כאשר החלק הראשון כבר איננו קיים. מכיון שישנם מהדרים שבודקים את שני החלקים בכל מקרה, מומלץ להשתמש בשיטה שמופיעה בגוף התכנית ולא בדרך מקוצרת לכאורה.

(34)-(29) יוצאים מן הלולאה כאשר מוצאים את השם של החבר ברשימת השמות (NIMTZA=TRUE, השם נמצא במקום I ברשימה). אפשרות נוספת לצאת מהלולאה היא לאחר שסורקים את כל הרשימה ולא מוצאים את שם החבר (NIMTZA=FALSE, I שווה ל: GODEL_RESHIMA+1). לכן מדפיסים את המספר של החבר אם NIMTZA=TRUE, אחרת מודיעים על כך שהוא איננו נמצא ברשימה.

תהליך החיפוש הליניארי הוא פשוט, אך לא יעיל. אם יש N איברים במערך, מספר ההשוואות שנצטרך לבצע עד שנמצא את האיבר הרצוי הוא 1 במקרה הטוב ביותר (כאשר האיבר שמחפשים נמצא בראש הרשימה), או N במקרה הגרוע ביותר (אם האיבר שמחפשים נמצא בסוף הרשימה). מספר ההשוואות הוא N/2 בממוצע.

<u>7.2.2 חיפוש בינרי</u>

שיטה הרבה יותר יעילה לחיפוש היא שיטת <u>החיפוש הבינרי</u> (Binary Search). יש הגבלה לשיטה זו, כי אפשר להשתמש בה רק על מערך ממוין, אשר יכול להיות בסדר מיון עולה או בסדר מיון יורד. חיפוש ליניארי, כפי שראינו קודם, מתאים גם לרשימה לא ממויינת.

חיפוש בינרי דומה לשיטת חיפוש במילון או במדריך טלפון. פותחים את הספר לאמצעו. אם למזלנו פתחנו אותו בדף הנכון, מחפשים את המלה ומסיימים. אם הערך של המלה שמחפשים קטן מן הערך של המלה שמופיעה בדף הפתוח לעינינו, עוברים לדף שנמצא באמצע החלק הראשון, ז"א בין התחלת הספר לבין הדף שפתחנו. אם הערך של המלה שמחפשים גדול מן הערך של המלה שמופיעה בראש הדף שפתחנו, עוברים לדף שבאמצע החלק השני של הספר. ממשיכים בשיטה זו, ז"א בחלוקה ל-2 של תחום הדפים שבו יש אפשרות למצוא את המלה המבוקשת, עד שנמצא את הדף שבו היא כתובה. בדוגמא זו מניחים סדר מיון עולה, אך אפשר לחפש בשיטה זו גם כאשר סדר המיון יורד.

לדוגמא נניח שמחפשים את המספר 16 ברשימה בת 7 איברים שנמצאים בסדר עולה:

הערכים:	4	9	11	13	16	18	27
מספר האיבר ברשימה:	1	2	3	4	5	6	7

הערך האמצעי ברשימה הוא 13. מכיון ש-16 גדול ממנו, עוברים לאמצע האיזור הימני שכולל את האיברים בעלי הערכים הגבוהים יותר "27, 18, 16" ושולפים

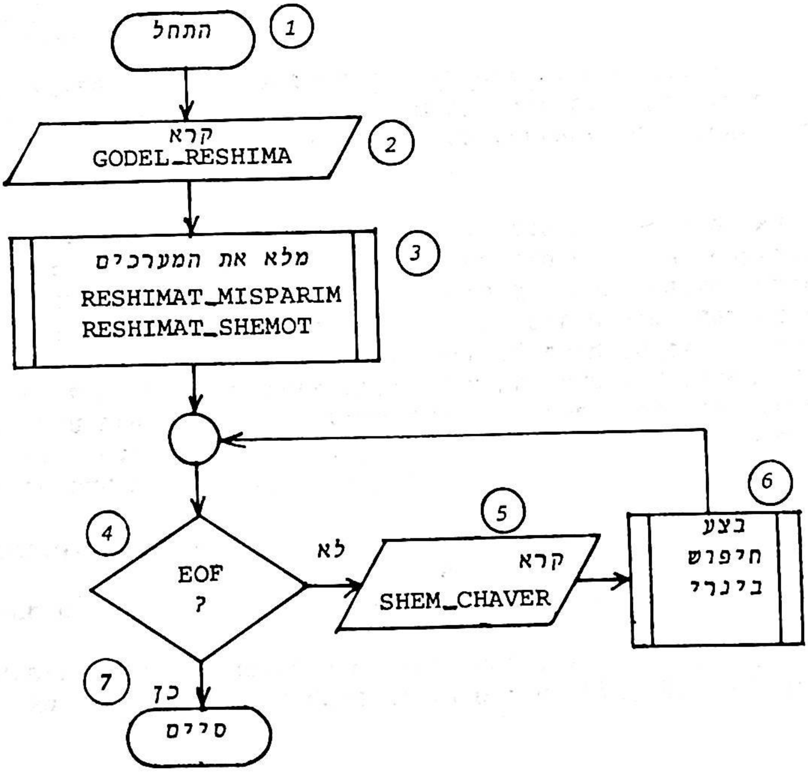
את המספר האמצעי 18. הוא גדול מן המספר המבוקש ולכן עוברים לחפש בין המספרים משמאל, שערכם קטן יותר. באיזור זה ישנו רק מספר אחד 16, שהוא המספר המבוקש והבדיקה מסתיימת. אילו מספר זה היה למשל 15, היינו צריכים להגיע למסקנה ש-16 לא נמצא ברשימה, כי עשינו את כל הצעדים האמורים להוביל אליו, אך ללא הצלחה.

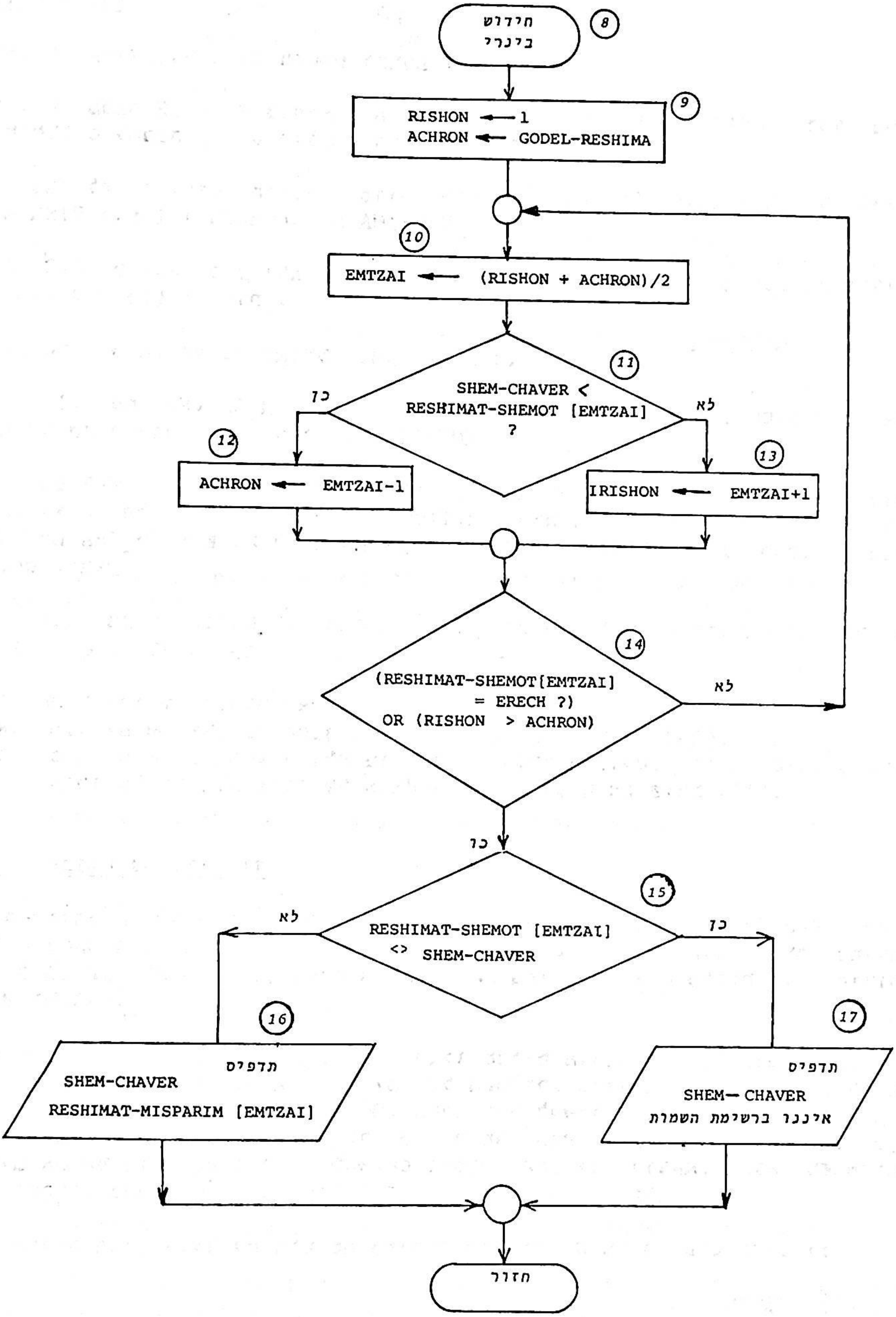
בדוגמא זו מצאנו את המספר המבוקש לאחר ההשוואה השלישית. זהו מספר ההשוואות המירבי שיש לעשות ברשימה של 7 איברים. ברשימה של 15 איברים, יש צורך לבצע עד 4 השוואות. זוהי השוואה אחת יותר מאשר בדוגמא הקודמת, כי לאחר החלוקה הראשונה יש שתי רשימות של 7 איברים. בצורה כללית, המספר המירבי של השוואות מתואר בטבלה למטה:

מספר איברים	מספר מירבי של השוואות
1	1
2, 3	2
4, 5, 6, 7	3
$2^{x-1} \ldots 2^{x}-1$	x

אם $N=2^{x-1}$, נקבל ש: $x-1=\log_2 N$, או $x=\log_2 N+1$. x נשאר ללא שינוי אע"פ ש-N גדל, עד שנגיע ל: $N=2^x$. בנקודה זו $\log_2 N$ גדול ב-1 מערכו כאשר $N=2^{x-1}$. נוכל לסכם ולומר ש: $x=TRUNC(\log_2 N)+1$ הוא המספר המירבי של ההשוואות בחיפוש בינרי במערך של N איברים.

נתאר עתה את האלגוריתם לביצוע חיפוש בינרי עבור דוגמת התכנות הקודמת למציאת שם במדריך הטלפון.




חידוש
בינרי
8
RISHON ← 1
ACHRON ← GODEL-RESHIMA
9
10
EMTZAI ← (RISHON + ACHRON)/2
11
SHEM-CHAVER <
RESHIMAT-SHEMOT [EMTZAI]
?
כן
לא
12
ACHRON ← EMTZAI-1
13
IRISHON ← EMTZAI+1
14
(RESHIMAT-SHEMOT[EMTZAI]
= ERECH ?)
OR (RISHON > ACHRON)
לא
כן
15
RESHIMAT-SHEMOT [EMTZAI]
<> SHEM-CHAVER
לא
כן
16
תדפיס
SHEM-CHAVER
RESHIMAT-MISPARIM [EMTZAI]
17
תדפיס
SHEM— CHAVER
איננו ברשימת השמות
חזור

<u>הסבר לתרשים:</u>

(3)-(2) קריאה של מערך השמות והמספרים.

(6)-(4) מבנה DOWHILE לקריאת שמות חברים וחיפוש שמם ברשימה. בכל פעם מסתעפים למבנה (6) שמכיל את החיפוש הבינרי.

(9) בתחילת החיפוש הבינרי טווח החיפוש הוא כל האיברים במערך. לכן, RISHON הוא 1 ו-ACHRON הוא GODEL_RESHIMA.

(14)-(10) לולאה של פקודת DOUNTIL, שכוללת בתוכה את הפקודה IF. הפעולות (10) עד (13) יכולות להיות קטע נפרד.

(10) מוצאים את האיבר האמצעי במערך הנתון.

(12)-(11) אם SHEM_CHAVER קטן מן הערך של האיבר האמצעי, חשוב לדעת את הערך של האיבר שנמצא לפניו, EMTZAI-1.

(13) אם SHEM_CHAVER גדול מן הערך של האיבר האמצעי, עוברים לקבוצה העליונה של האיברים והערך של RISHON יהיה גדול ב-1 מהערך של EMTZAI. אם SHEM_CHAVER שווה לערך האמצעי, אין צורך ב-(13), אבל הפקודה בכל זאת תתבצע.

(14) סוף מבנה DOUNTIL. מחזירים את הביצוע להתחלת הלולאה אם עוד לא הגענו למצב של הכרעה.

(15) אפשר להגיע להכרעה כאשר:
א. מוצאים את שמו של החבר ברשימה ומבצעים את שורה (14).
ב. סורקים את כל המערך ולא מוצאים את הערך המבוקש. הדבר קורה כאשר ערכו של RISHON עולה על ACHRON ואז מבצעים את שורה (17).

<u>7.2.3 תכנות בפסודו-קוד</u>

את האלגוריתם של החיפוש אפשר לכתוב בשיטת <u>פסודו-קוד</u> (Pseudo Code), אשר מחליפה במקרים רבים את תרשימי הזרימה. הכתיבה של פסודו-קוד דומה במידה רבה לתכנות, אלא שחלק מן הפרטים והלוגיקה כתובים בצורה כללית ולא בפירוט הדרוש בתכנית.

בתרשימי זרימה יש לעתים קושי גרפי להציג מבנים מורכבים ולכן כותבים קטעים נפרדים בכדי להציג חלקים שונים של התהליך. בדוגמא הקודמת למשל, היתה לולאה של DOUNTIL בתוך IFTHENELSE, אשר גרמה לקשיי שרטוט והבנה. יש יחס חד-חד-ערכי בין הפסודו-קוד ותרשימי זרימה, אלא שבמקום לצייר את המבנים, כותבים את שמותיהם. אין צורך להשתמש בשמות המבנים דווקא, אלא משתמשים ברוב המקרים במלים אשר מתאימות לשפת התכנות שבה מתכנתים.

להלן תכנית פסודו-קוד לחיפוש שם במדריך הטלפון בשיטת החיפוש הבינרי:

```
(1)  BEGIN
(2)     READ GODEL_RESHIMA;
```

```
(3)      READ RESHIMAT_SHEMOT;
(4)      READ RESHIMAT_MISPARIM;
(5)        WHILE NOT EOF DO
(6)          BEGIN
(7)              READ SHEM_CHAVER;
(8)              RISHON <-- 1;
(9)              ACHRON <-- GODEL_RESHIMA;
(10)             REPEAT
(11)                EMTZAI <-- (RISHON + ACHRON) / 2;
(12)                IF SHEM_CHAVER < RESHIMAT_SHEMOT[EMTZAI] THEN
(13)         <-- התעלם מהערכים העליונים ע"י עדכון הערך של ACHRON
(14)                ELSE
(15)         <-- התעלם מהערכים התחתונים ע"י עדכון הערך של RISHON
(16)             UNTIL           סיום הלולאה;
(17)             IF RESHIMAT_SHEMOT[EMTZAI] <> SHEM_CHAVER THEN
(18)                   כתוב שהשם המבוקש לא נמצא
(19)             ELSE
(20)               כתוב את שם החבר ומספרו הסידורי
(21)         END
(22)         END
```

לאחר הכתיבה של האלגוריתם בשיטה זו מחליפים את הפעולות (13), (15), (16), (18), (20) בפקודות פסקל אשר מקבילות להן. בדרך זו מקבלים שלד של תכנית בשפת תכנות. עתה יש צורך להשלים ולסיים את התכנית כנדרש. בהמשך הספר הצגנו אלגוריתמים בשיטת תרשימי זרימה בלבד, אך בפתרון התרגילים התלמיד יוכל לבחור בין שיטת תרשימי זרימה לבין פסודו-קוד.

נשבץ עכשיו תהליך של חיפוש בינרי לתכנית הקודמת MISPAREI_TELEPHONE. יש להגדיר בה שלושה משתנים נוספים: RISHON, ACHRON, EMTZAI: 1..100.

במקום שורות (22)-(33) באותה תכנית, נכתוב עכשיו את הפקודות הבאות:

```
RISHON:= 1;
ACHRON:= GODEL_RESHIMA;
REPEAT
   EMTZAI:= (RISHON+ACHRON) DIV 2;
   IF SHEM_CHAVER < RESHIMAT_SHEMOT[EMTZAI] THEN
     (* חפש בקטע התחתון *)
     ACHRON:= EMTZAI-1
   ELSE    (* חפש בקטע העליון *)
     RISHON:= EMTZAI+1
UNTIL (RESHIMAT_SHEMOT[EMTZAI]= SHEM_CHAVER) OR (RISHON > ACHRON);
IF RESHIMAT_SHEMOT[EMTZAI] <> SHEM_CHAVER THEN
   WRITELN (SHEM_CHAVER, 'אין אדם בשם^')
ELSE
   WRITELN (SHEM_CHAVER, '^':10, RESHIMAT_MISPARIM[EMTZAI];
```

היה אפשר להרחיב את הדוגמא גם למקרה של אינדקסים מסוג CHAR. לשם כך יש לכתוב: RISHON:=SUCC[EMTZAI] וגם: ACHRON:=PRED[EMTZAI], במקום לכתוב: RISHON:=EMTZAI+1 וגם: ACHRON:=EMTZAI-1.

המספר הממוצע של השוואות בחיפוש ליניארי הוא N/2 ובחיפוש בינרי מספר ההשוואות הוא $TRUNC(log_2{}^N)+1$. נובע מזה שכאשר N>8, מספר ההשוואות בחיפוש בינרי קטן ממספר ההשוואות בחיפוש ליניארי. אולם, בביצוע לולאה של חיפוש בינרי יש יותר פעולות מאשר בביצוע לולאה של חיפוש ליניארי ולכן גבול הכדאיות איננו נקבע לפי התנאי N>8 בלבד.

על פי הניסיון, מומלץ להשתמש בחיפוש בינרי כאשר יש רשימה ממויינת בעלת יותר מ-20 איברים. בכל מקרה אחר החיפוש הליניארי עדיף.

7.3 מיון

מיון (Sort) הוא הפעולה של סידור איברים לפי תכונה מוגדרת, כמו למשל לפי הערך שלהם. אפשר למיין בשני סדרי מיון (Collating Sequences):

א. סדר עולה (Ascending Order) - לאחר המיון, האיבר בעל הערך הקטן ביותר יהיה הראשון ברשימה והאיבר בעל הערך הגדול ביותר יהיה האחרון ברשימה.

ב. סדר יורד (Descending Order) - לאחר המיון, האיבר בעל הערך הגדול ביותר יהיה הראשון והאיבר בעל הערך הקטן ביותר יהיה האחרון ברשימה.

בהמשך נציג שלוש דרכים למיון של נתונים, אשר ערוכים במערך חד-מימדי (וקטור) A שמוגדר כך:

```
A:ARRAY[1..N] OF ARRAY_TYPE
```

7.3.1 מיון ע"י סריקה (Insertion Sort)

מניחים שהערכים המשובצים כבר בווקטור הם ממויינים. את הערך החדש צריך להוסיף לוקטור במקום המתאים לפי סדר המיון. אם הוכנסו כבר K ערכים לוקטור A, אפשר להכניס לוקטור בדרך זו את האיבר החדש X, שמספרו הסידורי K+1. הפעולות הדרושות הן:

א. מוצאים את המקום J, כך ש: X < A[J].
ב. מזיזים את האיברים ממקום J ואילך למקום הבא בווקטור.
ג. מכניסים את X למקום J.

התכנות לביצוע השלב הראשון:

```
MCHAPES:= TRUE;
J:= 1;
WHILE (J <= K) AND MCHAPES DO
   IF X < A[J] THEN
      MCHAPES:= FALSE
   ELSE
      J:= J + 1;
```

אם X איננו קטן מאיבר כלשהוא בתחום A[1] עד A[K], הערך של J יגיע לערך K+1. המסקנה תהיה שיש להכניס את האיבר החדש X למקום K+1 ש-J מצביע עליו, כלומר בסוף הוקטור.

התכנות לביצוע השלב השני:

```
FOR I:= K DOWNTO J DO
    A [I+1]:= A[I];
```

אם X איננו קטן מאיבר כלשהוא במערך A, המשתנה J יהיה שווה ל: K+1 ופקודת FOR זו לא תתבצע. כלומר, אם X גדול מכל ערך בווקטור, אין צורך להזיז אף איבר שהוא.

התכנות לביצוע השלב השלישי:

```
A[J]:= X;
```

כל סריקה בווקטור A נקראת מעבר (Pass). נסתכל עכשיו על תוצאות המעברים במיון הרשימה שמופיעה למטה. כוכבית מסמנת את האיבר הנוסף בכל מעבר.

רשימה מקורית	אחרי המעבר הראשון	אחרי המעבר השני	אחרי המעבר השלישי	אחרי המעבר הרביעי	אחרי המעבר החמישי	אחרי המעבר השישי	אחרי המעבר השביעי
10	10*	10	8*	1*	1	1	1
12		12*	10	8	6*	6	4*
8			12	10	8	8	6
1				12	10	10	8
6					12	12	10
19						19*	12
4							19

למיון הרשימה בת 7 איברים נדרשו 7 מעברי מיון, כי בכל מעבר נוסף איבר אחד לרשימה הממויינת.

7.3.2 מיון ע"י בחירה ליניארית (Linear Selection)

נניח שישנם N ערכים לא מסודרים בווקטור. נבצע את הצעדים הבאים בכדי למיין אותם:

א. נציב 1 במשתנה R, אשר מיועד להצביע על הערך הקטן ביותר בווקטור. נשווה את A[R] אל A[2] ונקבל שתי תוצאות אפשריות להשוואה זו:
- A[R] >= A[2]. R ישאר 1.
- A[R] < A[2], R יקבל את הערך 2.

ב. נשווה את A[3] אל A[R]. אם A[R] < A[3], אז R יהפוך להיות 3, אחרת הוא ישאר ללא שינוי.

ג. נמשיך כך עבור A[4], A[5] וכו', עד שנשווה את A[N] אל A[R]. לאחר כל השוואה, האינדקס R יצביע על האיבר שערכו קטן יותר. לאחר ההשוואה האחרונה האינדקס יצביע על האיבר הקטן ביותר והמשתנה A[R] יכיל את הערך הקטן מבין כל K הערכים בווקטור.

ד. עתה יש להחליף את A[1] ב-A[R]. עכשיו האיבר בעל הערך הקטן מכל האיברים נמצא במקום הראשון בווקטור. כך מסתיים המעבר הראשון.

המעבר השני דומה לראשון, אלא שמציבים 2 במשתנה R. עורכים את החיפוש מהאיבר A[3] עד האיבר A[N], כדי למצוא את האיבר הקטן ביותר מבין N-1 איברים שאחרי האיבר הראשון. כזכור, האיבר הראשון נבחר במעבר הראשון כבעל הערך הקטן ביותר. האיבר הקטן ביותר שנמצא עכשיו יוצב לפי ערכו במקום השני במערך ויהיה השני בסדר המיון.

כך ממשיכים במעברים, עד אשר מגיעים לכך שכל האיברים מסודרים כנדרש.
התכנות עבור מיון בשיטה זו:

```
(1)    FOR I:=1 TO N-1 DO
(2)       BEGIN
(3)          R:=I;
(4)          FOR J:= I+1 TO N DO
(5)            IF A[J] < A[R] THEN
(6)              R:=J;
(7)          IF I <> R  THEN
(8)              BEGIN
(9)                ZMANI:= A[I];
(10)               A[I]:= A[R];
(11)               A[R]:= ZMANI
(12)             END
(13)      END;
```

<u>הסבר:</u>

(1) נשווה כל איבר, מ-A[1] עד A[N-1] אל קבוצת האיברים הבאים אחריו.

(3) במצב התחלתי מניחים שהקטן מבין כל האיברים העוקבים נמצא במקום I.

(4) קבוצת האיברים שבאים אחרי A[I] כוללת את A[I+1] עד A[N].

(6) המשתנה R מסמל את האינדקס של האיבר הקטן ביותר שטופל עד עכשיו. לכן, המשתנה R יקבל את ערכו של J אם A[J]<A[R]. בסוף הבדיקה של שורות (4)-(6), R יהיה האינדקס של האיבר הקטן ביותר בכל התחום (ז"א מ-A[I] עד ל- A[N]).

(7)-(12) יש תמיד אפשרות שהאיבר שהיה במקום I הוא הקטן ביותר מתוך האיברים שנשארו ולכן אין צורך בהחלפה כלשהי. אם נקבל I<>R, נובע ש-I לא היה האיבר הקטן ביותר ולכן נחליף אותו באיבר R כדי שהאיבר הקטן ביותר יהיה במקום I. במעבר הראשון I=1, אח"כ I=2 וכן הלאה עד N-1, כאשר מסתיימים מעברי המיון.

לדוגמא נציג מיון בשיטה זו:

רשימה מקורית	אחרי המעבר הראשון	אחרי המעבר השני	אחרי המעבר השלישי	אחרי המעבר הרביעי	אחרי המעבר החמישי	אחרי המעבר השישי
10	1	1	1	1	1	1
12	12	4	4	4	4	4
8	8	8	6	6	6	6
1	10	10	10	8	8	8
6	6	6	8	10	10	10
19	19	19	19	19	19	12
4	4	12	12	12	12	19

7.3.3 מיון ע"י החלפה (Exchange Sort), או מיון בועות (Bubble Sort)

תיאור השיטה:

א. במעבר הראשון משווים את האיבר הראשון אל השני ומחליפים ביניהם, אם השני קטן מן הראשון. עתה משווים את האיבר השני (החדש) אל השלישי ומחליפים אם דרוש. כך ממשיכים עד שמשווים את איבר (N-1) אל איבר N. בדרך ההשוואה וההחלפה "יתגלגל" (ז"א יעלה כמו בועה) האיבר הגדול ביותר אל סוף הרשימה. כותבים את האינדקס של האיבר הראשון בזוג האחרון שהחלפנו ומעבירים אותו למשתנה J. מן האיבר השני בזוג האחרון שהשתתף בהחלפה ועד סוף הווקטור כל האיברים כבר מסודרים ואין צורך לבדוק אותם במעברים הבאים.

ב. במעבר השני שוב משווים את האיבר הראשון עם השני, השני עם השלישי, וכו', עד להשוואה האחרונה שהיא השוואת איבר (J-1) אל איבר J.

ג. בכל שלב מספר ההשוואות יורד לפחות ב-1, כי מן האיבר השני בזוג האחרון שהחלפנו לאחרונה ואילך - הווקטור מסודר.

ד. התהליך מסתיים כאשר יש מעבר שבו אין אף החלפה.

דוגמא של מיון ע"י החלפה:

מעבר ראשון מקורי J=4	לאחר החלפה ראשונה	לאחר החלפה שנייה	לאחר החלפה שלישית	סוף מעבר ראשון
7	6	6	6	
6	7	5	5	
5	5	7	4	
4	4	4	7	

מעבר שני J=3	לאחר החלפה ראשונה	לאחר החלפה שנייה	
			סוף מעבר שני
6	5	5	
5	6	4	
4	4	6	
7	7	7	

מעבר שלישי J=2	לאחר החלפה ראשונה	
		סוף מעבר שלישי
5	4	
4	5	
6	6	
7	7	J=1 סוף המיון

שימו לב לכך שהערך של J ירד ב-1 בכל מעבר. אם הווקטור היה יותר מסודר, או ממויין, אז הערך של J יכול היה לרדת עוד יותר ממעבר למעבר.

דוגמא נוספת:

מעבר ראשון סידור מקורי J=4	לאחר החלפה ראשונה	לאחר החלפה שנייה	לאחר החלפה שלישית	מעבר שני J=3
7	4	4	4	4
4	7	5	5	5
5	5	7	6	6
6	6	6	7	7
			סוף מעבר ראשון	J=0 סוף המיון

בדוגמא זו, J ירד במעבר השני מ-3 ל-0 ולכן אין צורך במעבר נוסף.

תכנית מיון בשיטת הבועות:

```
J:=N;
WHILE   J <> 0 DO
  BEGIN
    L:=0;
    FOR  I:=1 TO J-1 DO
      IF A[I] > A[I+1] THEN
         BEGIN
            ZMANI:= A[I];
            A[I]:= A[I+1];
            A[I+1]:= ZMANI;
            L:=I
         END;
    J:=L
  END;
```

<u>הסבר:</u>

(1) בווקטור המקורי יש N איברים.

(6)-(12) השוואת הזוגות. המשתנה L מקבל את הערך של האיבר הראשון בזוג האחרון שנבדק.

(13) המשתנה J מקבל את הערך הסופי של L. בפעם הבאה הזוג האחרון שנבדוק יתחיל מאיבר J-1. המשתנה L יקבל ערך חדש של 0 ורק ישתנה במהלך ההחלפות שתתבצענה כתוצאה מפקודת IF בשורה (6).

7.3.4 השוואה בין שיטות המיון (*)

א. מיון ע"י סריקה

בשיטה זו מספר ההשוואות קטן יותר מאשר בשיטות האחרות ובממוצע דרושות לו $N^2/4$ השוואות. אפשר להקטין את מספר ההשוואות אם משתמשים בשיטת חיפוש בינרי במקום בשיטת חיפוש ליניארי בכדי למצוא את המקום של האיבר שנבדק. בדרך זו מספר ההשוואות מתקרב ל-$N*\log_2 N$.

במיון ע"י סריקה מספר ההחלפות גדול. ככל שהרשימה יותר ארוכה, מספר ההחלפות יותר גדול.

ב. מיון ע"י בחירה ליניארית

במעבר הראשון משווים את האיבר הראשון עם N-1 האיברים האחרים. במעבר השני משווים את האיבר השני עם N-2 האיברים הבאים אחריו. במעבר השלישי משווים את האיבר השלישי עם N-3 האיברים הבאים אחריו וכו'. מספר ההשוואות שווה ל: $(N-1)+(N-2)+.....+2+1$. זהו טור אריתמטי שסכומו הוא:

$$(\text{ראשון} + \text{אחרון})(\frac{\text{מספר האיברים}}{2}) = (\frac{N-1}{2})[(N-1)+1] = \frac{(N-1)N}{2}$$

במעבר I מחליפים את A[I] בקטן מבין כל האיברים שבאים אחריו. יש אם-כן, החלפה אחת בכל מעבר ולכן יש N-1 החלפות בכל N-1 המעברים. בשיטה זו מספר ההשוואות זהה, גם אם הווקטור המקורי מסודר במלואו או בחלקו.

ג. מיון ע"י החלפה, או מיון בועות

שיטה זו אינה יעילה יותר מבחינה מתמטית מאשר מיון ע"י בחירה לינארית, אך היא יותר מורכבת ממנה. המספר המירבי של מעברים הוא N-1 מכיון שלאחר כל מעבר, לפחות זוג אחד נוסף כבר מסודר. גם בשיטה זו, כמו במיון ע"י בחירה לינארית, יש N-1 השוואות במעבר הראשון, N-2 במעבר השני, עד שיש השוואה אחת בלבד במעבר האחרון. לכן, מספר ההשוואות הוא $N(N-1)/2$.

ברוב המקרים אין צורך לבצע את כל N-1 המעברים. בדרך כלל, המעברים שאין צורך לבצע הם בסיום התהליך כאשר מספר ההשוואות קטן ולכן החיסכון איננו גדול. הבעיה העיקרית בשיטה זו היא מספר ההחלפות. הרצוי הוא, שתהיה החלפה אחת בכל מעבר, דהיינו N-1 בכל התהליך כמו במיון ע"י בחירה ליניארית. למעשה, מספר ההחלפות בממוצע הוא מחצית ממספר ההשוואות, כלומר $N(N-1)/4$ החלפות בכל התהליך.

בשיטת מיון זו כדאי להשתמש כאשר הנתונים כמעט מסודרים בתחילת העבודה. מספר המעברים וגם מספר ההשוואות וההחלפות יהיה יותר קטן.

ד. המלצה

- מיון ע"י סריקה טוב כאשר מספר האיברים קטן (למטה מ-20).
- מיון ע"י בחירה טוב אם הרשימה גדולה.
- מיון ע"י החלפה טוב אם הרשימה כמעט ממויינת ויש להחליף רק איברים מעטים שאינם לפי סדר המיון.

מן הראוי לציין שיש שיטות מיון מורכבות ויעילות יותר מאלו שהוזכרו כאן. השיטות שהוצגו הן הפשוטות והמקובלות לתכנות.

7.3.5 תכנית המיון

כסיכום לפרק שיטות מיון נציג לדוגמא תכנית מיון שלימה עם תרשימי זרימה.

נניח שקיימים הנתונים הבאים עבור כל פועל במפעל:
- שם.
- משכורת.
- אחד משלושת הציונים A, B, C אשר מגדירים את איכות עבודתו.

נתונים אלה אינם מסודרים. יש לקרוא אותם למחשב, לערוך את הפלט הנדרש ולהציג אותו.

הפלט הוא רשימת העובדים מסודרת (ממויינת) בסדר אלפבתי של שם העובד. הנתונים לכל עובד כוללים את שם העובד, המשכורת והציון. במקום קוד הציון יש להדפיס את הכינוי שלו: 'METZUYAN' במקום הקוד 'A', 'TOV' במקום 'B', ו-'MEMUZA' במקום 'C'.

תחילה יש לקבוע את שיטת המיון ולהכין תרשים זרימה. נקרא את הנתונים ונמיין אותם בעת הצבתם במערך שבזכרון. השיטה תהיה למצוא את המקום בעבור השם החדש במערך מסודר של שמות ולהציב אותו במקום זה, כלומר מיון ע"י סריקה.

המשתנים שיוגדרו בתוכנית:

MAXOVDIM:	מספר מקסימלי של עובדים שהתוכנית תוכל לטפל בו.
ORECH:	גודל המערך שמחזיק את השם של הפועל (מערך של תווים).
OVED_CHADASH:	מערך שמחזיק את השם של העובד שהתקבל כקלט.
TZIYUN_CHADASH:	הציון של העובד החדש.
SCHRR CHADASH:	השכר של העובד החדש.
MAKOM:	שם של אינדקס עבור המערך OVED_CHADASH. התחום הוא מ-1 עד ל-ORECH.
MISPAR_OVDIM:	מספר העובדים שקראנו עד עכשיו.
OVDIM:	מערך של שמות העובדים. כל איבר ב-OVDIM הוא בעצמו מערך שמכיל את השם של כל אחד מן העובדים.
TZIYUN:	מערך שמכיל באיבר I את הציון של עובד I. הציון הוא אחד האותיות 'A', 'B', או 'C'.
SCHAR:	מערך שמכיל באיבר I את השכר של עובד I.

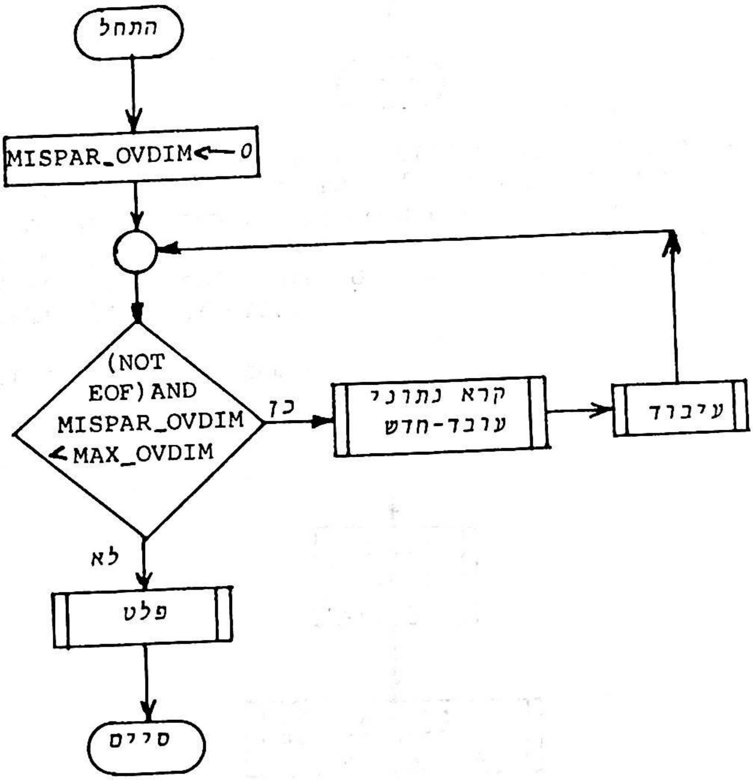
MISPAR: שם של אינדקס במערך OVDIM. התחום הוא מ-1 עד ל-MISPAR_OVDIM.

MCHAPES: משתנה בוליאני שערכו TRUE עד שמוצאים את המקום הנכון שבו יוכנס השם של הפועל הנוסף OVED_CHADASH.

MISPAR_CHADASH: המקום הנכון ברשימת העובדים עבור הפועל הנוסף ששמו נמצא במערך OVED_CHADASH.

תרשים זרימה לפתרון בעיה זו מוצג בהמשך.

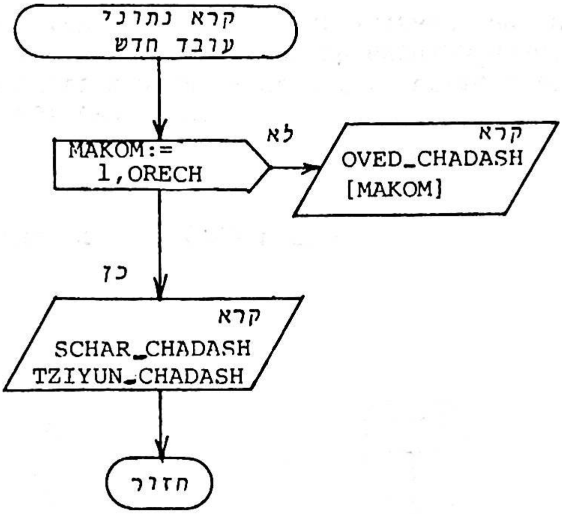
א. <u>תרשים זרימה ראשי</u> (שורות (18)–(54) בתכנית).



<u>הסבר:</u>

ממשיכים את הטיפול בנתונים כל עוד לא מגיעים לסוף הקובץ. מניחים שגודלו של הקובץ (MISPAR_OVDIM) לא עולה על גודל המערך (MAX_OVDIM) שהוקצה כדי להחזיק את שמות העובדים.

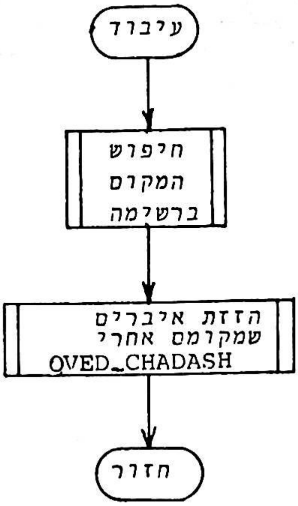
ב. קריאת נתונים של עובד חדש (שורות (22)-(25) בתכנית)



הסבר:

קוראים את האותיות שמרכיבות את השם של העובד הנוסף ומכניסים אותן למערך SHEM_CHADASH. קריאה זו נעשית בתוך לולאת FOR..DO בתוכנית. כדי להמנע מסיבוך יתר של התרשים, השתמשנו במבנה חדש שאיננו סטנדרטי, אבל נוח למי שמתכנת בפסקל את הפקודה FOR..DO. בתוך המבנה רשום שם האינדקס וגבולו התחתון והעליון. אחר-כך קוראים את המשתנים SCHAR_CHADASH ו-TZIYUN_CHADASH עבור אותו עובד.

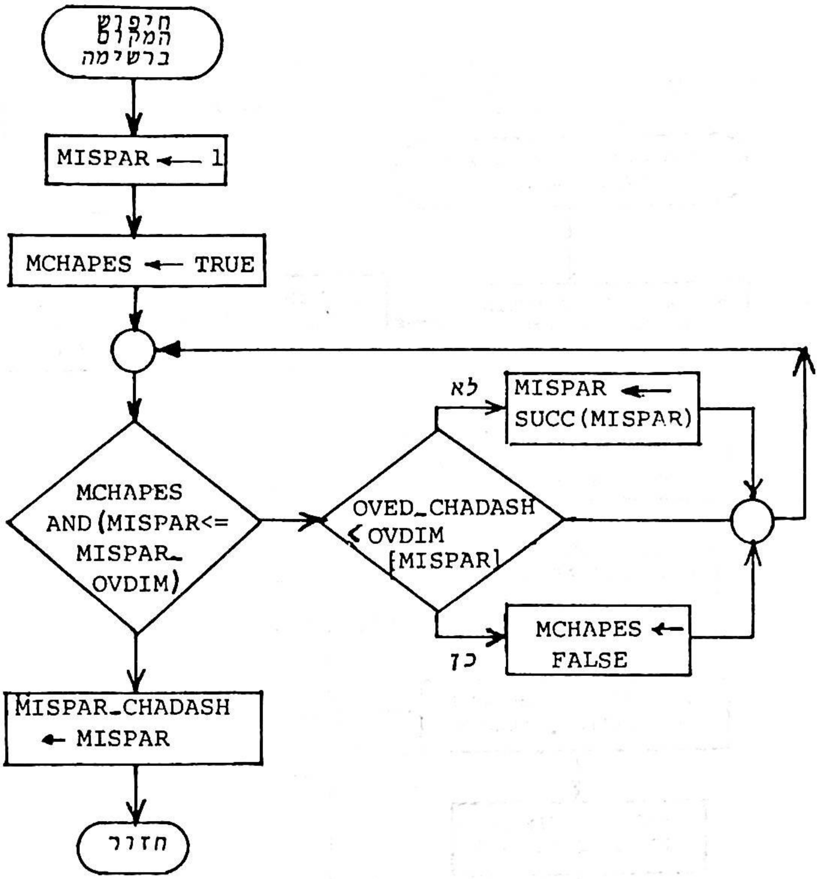
ג. עיבוד (שורות (26)–(43) בתכנית)



הסבר:

שיטת המיון היא חיפוש המקום המתאים ברשימת השמות עבור הנתון OVED_CHADASH שעכשיו נקרא, ז"א שמבצעים מיון ע"י סריקה. כאשר מוצאים את המקום המתאים, מזיזים במקום אחד את השמות שרשומים במקום זה ואילך אל כיוון סוף הרשימה. כך מתפנה מקום עבור השם החדש.

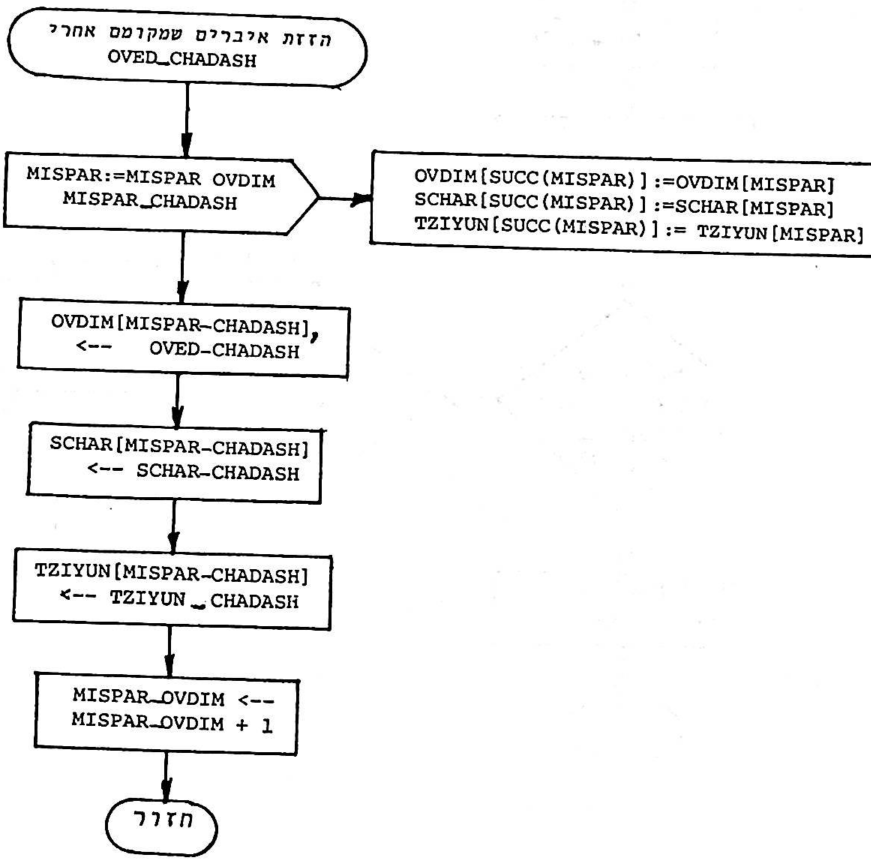
ד. <u>חיפוש מקום ברשימה עבור השם החדש</u> (שורות (26)-(33) בתכנית)



<u>הסבר:</u>

החיפוש נעשה ע"י השוואה של OVED_CHADASH אל כל אחד מן השמות ברשימה, עד אשר מוצאים שם שערכו האלפביתי יותר גדול ממנו. שם זה נמצא במקום שכינויו MISPAR_CHADASH וצריך להחליף אותו ע"י OVED_CHADASH. כאשר קולטים את השם של העובד הראשון אין צורך לעשות חיפוש. במקרה זה MISPAR (שערכו ההתחלתי הוא 1) גדול מ-MISPAR_OVDIM (שערכו עדיין 0) ולכן לא נכנסים כלל ללולאה.

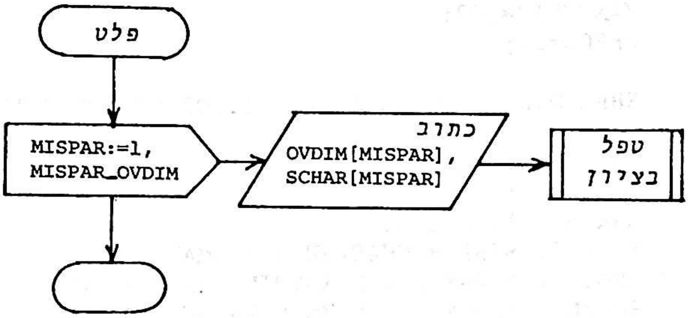
ה. הזזת איברים שמקומם אחרי OVED CHADASH
(שורות (34)-(43) בתכנית).



הסבר:

מזיזים כל שם שמקומו שווה למספר הסידורי MISPAR_CHADASH או למספר סידורי יותר גדול. רושמים OVED_CHADASH במקום שהתפנה ע"י הזזת OVDIM[MISPAR_CHADASH] ומוסיפים 1 ל-MISPAR_OVDIM. מבצעים פעולות מקבילות במערכים עבור שכר וציונים. כאשר מטפלים בעובד הראשון אין צורך בהזזה כי MISPAR_OVDIM עדיין שווה ל-0 ו-MISPAR_CHADASH שווה ל-1. לא תתבצע כל הזזה ולא נכנס כלל ללולאה.

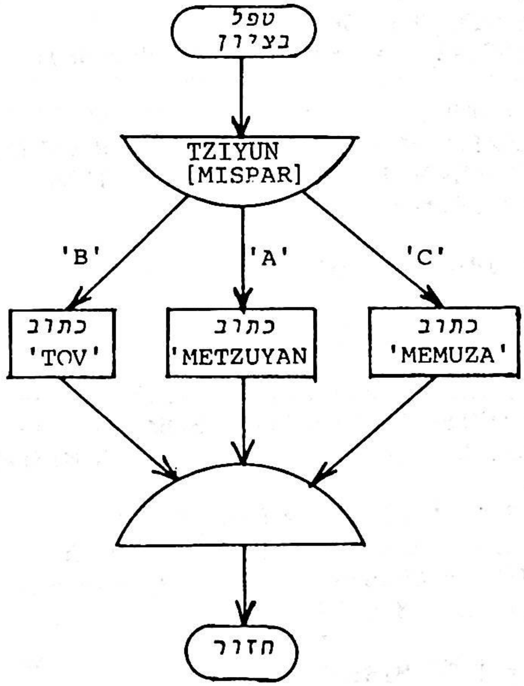
ו. הפקת הפלט (שורות (45)-(53) בתכנית)



הסבר:

כותבים את השם ואת השכר של כל עובד.

ז. טיפול בציון (שורות (48)-(52) בתכנית).



מדפיסים את שם הציון ולא את הקוד שלו.

ח. תכנית המיון

```
PROGRAM MASKORET13 (INPUT, OUTPUT);
CONST
          MAXOVDIM=100;
          ORECH=25;
TYPE
          SHEM_OVED= PACKED ARRAY [1..ORECH] OF CHAR;
VAR
          MISPAR_OVDIM: 0..MAXOVDIM;
          MCHAPES : BOOLEAN;
          MAKOM : 1..ORECH;
          MISPAR, MISPAR_CHADASH : 1..MAXOVDIM;
          OVDIM : ARRAY [1..MAXOVDIM] OF SHEM_OVED;
          SCHAR : ARRAY [1..MAXOVDIM] OF REAL;
          TZIYUN_CHADASH : 'A'..'C';
          TZIYUN : ARRAY [1..MAXOVDIM] OF 'A'..'C';
          OVED_CHADASH : SHEM_OVED;
          SCHAR_CHADASH : REAL;
BEGIN
  MISPAR_OVDIM:=0;
  WHILE NOT EOF AND (MISPAR_OVDIM < MAXOVDIM) DO
   BEGIN
     WRITELN ('הקש שם, שכר וציון');
     FOR MAKOM:= 1 TO ORECH DO
       READ (OVED_CHADASH[MAKOM]);
     READLN (SCHAR_CHADASH, TZIYUN_CHADASH);
     MISPAR:= 1;
     MCHAPES:= TRUE;
     WHILE MCHAPES AND (MISPAR <= MISPAR_OVDIM) DO
       IF OVED_CHADASH < OVDIM[MISPAR] THEN
          MCHAPES:= FALSE
       ELSE
          MISPAR:= SUCC(MISPAR);
     MISPAR_CHADASH:= MISPAR;
     FOR MISPAR:= MISPAR_OVDIM DOWNTO MISPAR_CHADASH DO
         BEGIN
           OVDIM[SUCC(MISPAR)]:= OVDIM[MISPAR];
           SCHAR[SUCC(MISPAR)]:= SCHAR[MISPAR];
           TZIYUN[SUCC(MISPAR)]:= TZIYUN[MISPAR]
         END;
     OVDIM[MISPAR_CHADASH]:= OVED_CHADASH;
     SCHAR[MISPAR_CHADASH]:= SCHAR_CHADASH;
     TZIYUN[MISPAR_CHADASH]:= TZIYUN_CHADASH;
     MISPAR_OVDIM:= MISPAR_OVDIM+1
   END;
  FOR MISPAR:= 1 TO MISPAR_OVDIM DO
   BEGIN
     WRITE (OVDIM[MISPAR]: 15, SCHAR_CHADASH:10:2);
     CASE TZIYUN[MISPAR] OF
       'A':WRITELN ('METZUYAN':10);
```

```
(50)          'B':WRITELN ('TOV':5);
(51)          'C':WRITELN ('MEMUTZA':8)
(52)        END
(53)      END
(54)  END.
```

<u>הסבר:</u>

(25)-(22) קוראים את אותיות השם, תו אחר תו בעזרת לולאה. מכיון ש-ORECH שווה ל-25, פירוש הדבר הוא ש-25 התווים הראשונים שיקראו ישתייכו ל-OVED_CHADASH.

אחרי השם אפשר להשאיר מספר כלשהוא של רווחים עד שכותבים מספר, שיתקבל כערכו של SCHAR_CHADASH. <u>מיד</u> אחרי המספר צריך להופיע הערך של TZIYUN_CHADASH, ז"א אחד מן הערכים 'A', 'B' או 'C'. אם נשאיר רווח אחד אחרי המספר, הרווח יתקבל כערכו של TZIYUN_CHADASH ויגרום לנתון שגוי.

רוב המהדרים מאפשרים לקרוא מחרוזת שלמה בלי צורך בלולאה. במהדר כזה, שיכול לקרוא PACKED ARRAY OF CHAR בלי לולאה, נוכל להחליף את השורות (23)-(25) בפקודה אחת:

```
READLN (OVED_CHADASH, SCHAR_CHADASH, TZIYUN_CHADASH)
```

(33)-(28) פקודות פסקל עבור הקטע "חיפוש מקום ברשימה עבור השם החדש". במקום השורה (32) אפשר לכתוב MISPAR:=MISPAR+1.

הפונקציות SUCC ו-PRED מוגדרות לא רק עבור סוגי נתונים המוגדרים ע"י המשתמש, אלא גם עבור הסוגים המוגדרים באופן אוטומטי:

- INTEGER - למשל, SUCC(10)=11,
- CHAR - למשל, SUCC('A')='B',
- BOOLEAN - למשל, SUCC(FALSE)=TRUE.

(44)-(34) תכנות הקטע "הזזת איברים שמקומם אחרי OVED_CHADASH". בפקודות (34)-(39) מזיזים את הנתונים של העובדים כדי לפנות מקום ברשימה עבור העובד החדש. בפקודות (40)-(43) מכניסים את פרטי העובד החדש למקום המתאים לו לפי סדר המיון.

<u>ט. דרך נוספת לפתרון הבעיה</u>

לכל עובד יש 3 סוגי נתונים. היה לכאורה אפשרי להכין טבלה שרוחבה 3 ואורכה כמספר העובדים (מערך דו-מימדי), אבל דרישת קדם למערך היא, שכל האיברים יוגדרו בצורה זהה. כאן, השדה "שם" (מסוג SHEM_OVED) שונה בסוגו מהשדה "שכר" (REAL) ומהשדה "ציון" (תת-טווח של CHAR). אופי הנתונים, אשר שונים במבנה שלהם, מחייב להגדיר שלושה מערכים שונים, כל אחד עם הגדרת הנתונים המתאימה לו. אפשרות נוספת היא להגדיר את הנתונים של כל פועל כרשומה. בנושא זה נדון בפרק 9.

בתכנית זו המיון נעשה בזמן קריאת הנתונים בשיטת הסריקה. היה אפשר לקרוא תחילה את כל הנתונים ואחר-כך למיין אותם בשיטת מיון ע"י החלפה, למשל.

בקטע התכנית הבאה מוצגת קריאה אקראית של נתוני עובדים למערכים OVDIM, SCHAR ו-TZIYUN מבלי למיין אותם לפי הסדר האלפביתי של שמות העובדים.

```
(1)  MISPAR_OVDIM:=0;
(2)  WHILE (NOT EOF) AND (MISPAR_OVDIM < MAXOVDIM) DO
(3)   BEGIN
(4)     MISPAR_OVDIM:= MISPAR_OVDIM+1;
(5)     WRITELN ('הקש שם, שכר, ציון');
(6)     FOR MAKOM:= 1 TO ORECH DO
(7)       READ (OVDIM [MISPAR_OVDIM] [MAKOM]);
(8)     READLN (SCHAR[MISPAR_OVDIM], TZIYUN[MISPAR_OVDIM])
(9)   END;
```

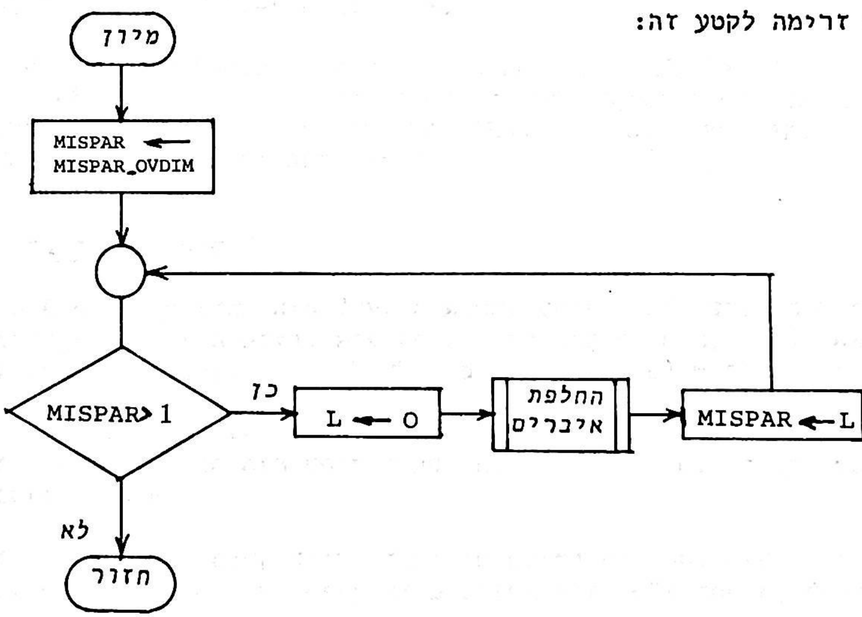
<u>הסבר:</u>

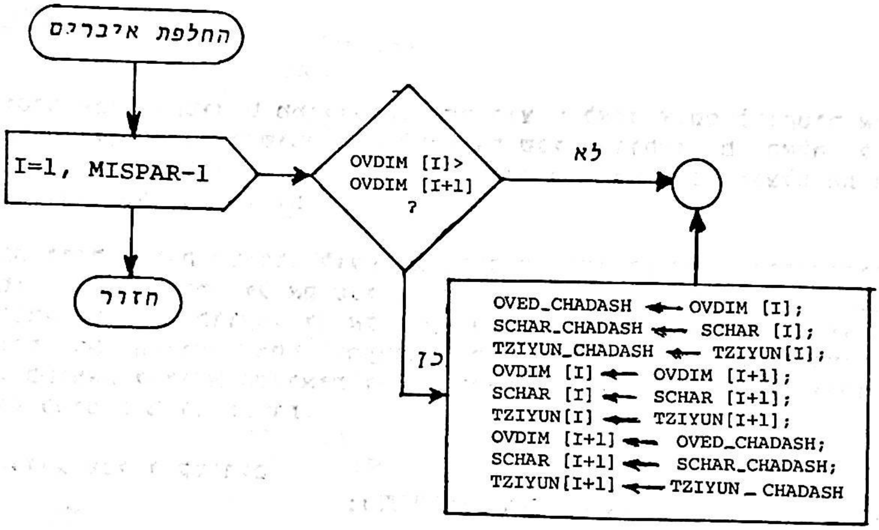
(7)-(6) קוראים כל שם למקומו במערך OVDIM. האיברים של המערך OVDIM הם בעצמם מערכים מסוג SHEM_OVED ולכן צריך לציין באופן מפורש את המקום בתוך המערך הזה שאליו קוראים את הנתונים. בפקודה זו רשומים שני אינדקסים, שכל אחד מהם כתוב בין סוגריים מרובעים.

ישנם מהדרים אשר מאפשרים לכתוב רק פקודה אחת במקום הפקודות שבשורות (6)-(7). פקודת הקריאה החדשה תהיה: READ (OVDIM[MISPAR_OVDIM])

כאשר רשימת השמות נמצאת במערך OVDIM, אפשר למיין אותה בשיטת ההחלפה. בתוך קטע המיון מבצעים באופן איטרטיבי את הקטע שנקרא "<u>החלפת איברים</u>". בסוף ביצוע פעולה של "החלפת איברים", רושמים בתוך המשתנה L את המספר של האיבר הראשון בזוג האחרון שהוחלף. בפעם הבאה תיעשה החלפה בין L האיברים הראשונים. אם אין צורך להחליף איברים, L ישאר עם ערכו המקורי 0 והתהליך של WHILE..DO יפסיק לפעול. גם אם נקבל L=1 בסיום של המעבר הקודם, אין צורך להמשיך כי לא נשארו זוגות איברים להשוואה.

תרשים זרימה לקטע זה:





הקידוד עבור קטע המיון ניתן להלן:

```
(1)   MISPAR := MISPAR_OVDIM;
(2)   WHILE MISPAR > 1 DO
(3)   BEGIN
(4)     L:=0;
(5)     FOR I:=1 TO MISPAR-1 DO
(6)       IF OVDIM[I] > OVDIM[I+1]
(7)        THEN
(8)         BEGIN
(9)           OVED_CHADASH := OVDIM[I];
(10)          SCHAR_CHADASH := SCHAR[I];
(11)          TZIYUN_CHADASH := TZIYUN[I];
(12)          OVDIM[I] := OVDIM[I+1];
(13)          SCHAR[I] := SCHAR[I+1];
(14)          TZIYUN[I] := TZIYUN[I+1];
(15)          OVDIM[I+1] := OVED_CHADASH;
(16)          SCHAR[I+1] := SCHAR_CHADASH;
(17)          TZIYUN[I+1] := TZIYUN_CHADASH;
(18)          L:= I
(19)        END;
(20)    MISPAR:=L
(21)  END;
```

פקודות (5)-(19) מתייחסות לקטע "החלפת איברים".

7.4 מיזוג

נניח שישנם שני וקטורים ממויינים, אשר רוצים לאחד אותם לווקטור אחד שגם הוא יהיה ממויין. הדרך הפשוטה היא לצרף את שני הווקטורים האלה לווקטור אחד ואח"כ למיין אותו. שיטה זו אינה יעילה כי היא אינה מנצלת את העובדה ששני הווקטורים כבר ממויינים.

שיטה הרבה יותר יעילה למיזוג לוקחת בחשבון את העובדה ששני הווקטורים כבר ממויינים: סורקים במקביל את שני הווקטורים ומעבירים את האיבר הקטן יותר מבין האיברים של שני הווקטורים אל וקטור שלישי, שהוא ווקטור היעד. שיטה זו מחייבת רק השוואה אחת והעברה אחת לכל איבר שיהיה בווקטור המטרה הממויין. כתוצאה יהיו N השוואות ו-N העברות, לעומת מספר הרבה יותר גדול של פעולות בשיטות מיון אחרות.

תכנית למיזוג שני וקטורים:

```
PROGRAM MIZUG (INPUT, OUTPUT);
VAR
  A, B, C: ARRAY[1..100] OF INTEGER;
  I, J, K, M, N, P: INTEGER;
BEGIN
    M:=0; N:=0;                    (* שתי פקודות בשורה אחת *)
    WRITELN ('הכנס את האיברים של וקטור A');
    WHILE NOT EOF DO
       BEGIN
         M:= M+1;
         READ (A[M])
       END;
    WRITELN  ('הכנס את האיברים של וקטור B');
    WHILE NOT EOF DO
       BEGIN
         N:= N+1;
         READ (A[N])
       END;
    I:=1;
    J:=1;
    K:=0;
    WHILE (I <= M) AND (J <= N) DO
         BEGIN
            K:= K+1;
            IF A[I] < B[J] THEN
               BEGIN
                 C[K]:= A[I];
                 I:=I+1
               END
            ELSE
               BEGIN
                 C[K]:= B[J];
                 IF A[I] = B[J] THEN
                   I:= I+1;
```

```
(35)                J:= J+1
(36)              END
(37)          END;
(38)      IF I <= M THEN
(39)          FOR P:=I TO M DO
(40)            BEGIN
(41)              K:=K+1;
(42)              C[K]:= A[P]
(43)            END
(44)      ELSE
(45)        IF J <= N THEN
(46)          FOR P:=J TO N DO
(47)            BEGIN
(48)              K:= K+1;
(49)              C[K]:= B[P]
(50)            END
(51)  END.
```

<u>הסבר</u>:

(4) המשתנים I, J, K יציינו את האינדקסים במערכים A, B, C בהתאמה.

(22)-(37) בוחרים את האיבר הבא ב-A או ב-B ומכניסים אותו ל-C.

(26)-(29) טיפול במצב שבו האיבר הבא ב-A קטן מהאיבר הבא ב-B. מעבירים את האיבר מ-A אל הוקטור החדש C.

(32) אם האיבר הבא ב-A איננו קטן מהאיבר הבא ב-B (כלומר האיבר ב-B קטן מן האיבר ב-A, או שווה לו), מעבירים את האיבר מ-B לוקטור החדש C.

(33)-(34) אם האיברים ב-A וב-B שווים, מעבירים ל-C רק איבר אחד. לכן, מקדמים את האינדקס של A מבלי להעביר את האיבר ל-C, כי את האיבר מ-B כבר העברנו בפקודה שבשורה (32). אם רוצים להעביר את שני האיברים צריך להשמיט את השורות (33) ו-(34). במקרה של שיוויון היינו מעבירים את האיבר שמקורו ב-B ואח"כ - את האיבר שמקורו ב-A.

(38)-(50) טיפול באיברים של אחד הווקטורים כאשר אין איברים נוספים בווקטור האחר.

(38)-(43) אם נשארו איברים ב-A, בשעה שב-B אין איברים נוספים, מעבירים אותם ל-C בזה אחר זה.

(44)-(50) אם ב-B נשארו איברים כאשר אין כבר איברים ב-A, מעבירים איברים אלה ל-C.

7.5 תרגילים

7.5.1 שאלות

(1) שנה את התכניות למיון לפי שלוש השיטות אשר רשומות בפרק זה, כדי שתבצענה מיון בסדר יורד, מגדול לקטן.

(2) שנה את השיטה שהוצגה למיון ע"י סריקה, בכדי שתמיין וקטור קיים עם ערכים (ולא כפי שמוצג בדוגמא).

(3) הרחב את התכנית למיזוג של שני וקטורים ממויינים בכדי שתמזג שלושה וקטורים ממויינים.

(4) (*) שנה את תכנית המיון לפי שיטת הבועות, כדי שתפעל בדרך הבאה:

א. תעביר את האיבר הגדול לסוף הרשימה (מקום n) ע"י השוואות, כמו שמוצג בפרק זה.
ב. תשווה את האיברים בזוגות, החל מן הזוג האחרון, שמורכב מן האיברים (n-1,n), עד הזוג הראשון שמורכב מן האיברים (1,2) ותעביר את האיבר הקטן ביותר למקום הראשון.
ג. תעביר את האיבר השני בגודלו למקום n-1, בדרך דומה ל-א'.
ד. תעביר את האיבר השני בקוטנו למקום 2, בדרך דומה ל-ב'.

ההשוואות וההעברות תמשכנה בדרך זו עד אשר נקבל וקטור ממויין.

(5) (*) הרחב את השיטה הקודמת כדי שכל מעבר יתחיל תמיד רק מהמקום של ההחלפה הקודמת, ויגיע רק עד למקום של ההחלפה שלפני האחרונה.

7.5.2 תכניות

(1) בארגון מסוים המשכורת המירבית הינה קבועה לפי שנות וותק, כפי שרשום בטבלה. אם יש יותר מ-25 שנות וותק, או פחות מ-3, אין חובה לנהוג לפי כללים אלו.

שנות וותק	משכורת מירבית
3-5	475.-
6-10	578.-
11-13	682.-
14-20	810.-
21-25	950.-

כתוב תכנית שמקבלת כקלט שמות של עובדים ושנות הוותק שלהם. הפלט יהיה דו"ח אשר יכיל את השמות של העובדים והמשכורת המירבית שלהם על פי שנות הוותק. בסיום יש להדפיס את מספר העובדים בכל קבוצת וותק.

(2) נתון וקטור של ערכים לא מסודרים. כתוב תכנית בכדי למצוא את החציון, אשר מוגדר כערך האמצעי של הווקטור. שלבי הפתרון:
- מסדרים את n האיברים בסדר עולה.
- אם n הוא אי-זוגי, החציון הוא הערך שמספרו הסידורי הוא (n+1)/2.

- אם n הוא זוגי, החציון הוא הממוצע של שני האיברים שמספריהם הסידוריים הם n/2, (n/2)+1.

(3) השמות של תלמידים בבי"ס מסוים מסודרים ברשימה ממויינת בסדר אלפביתי והשמות של התלמידים החדשים נמצאים ברשימה בלתי ממויינת. כתוב תכנית אשר קוראת את השמות של התלמידים החדשים, שמה אותם במערך ואח"כ קוראת את הרשימה של התלמידים שכבר לומדים בבית הספר. התכנית מדפיסה רשימה מעודכנת של כל התלמידים בזמן הקריאה של הרשימה הממויינת.

(4) חזור על תרגיל (3) כאשר שתי הקבוצות של שמות התלמידים אינן מסודרות בזמן הקלט.

(5) לתכנית המטפלת בהשבת אבידות יש שני סוגי קלט:
- תיאור המציאה, שם המוצא ומספר הטלפון שלו.
- שם המאבד ותיאור האבידה

אם יש התאמה בין תיאור האבידה לבין תיאור המציאה, התכנית מדפיסה שורה עם הפרטים הבאים:
תיאור האבידה, שם המאבד, שם המוצא ומספר הטלפון שלו.

כתוב תכנית אשר מסדרת את הקלט מן הסוג הראשון בסדר אלפביתי ומכניסה אותו למערך. כאשר היא מקבלת את הקלט מן הסוג השני היא עורכת חיפוש לינארי או בינרי במערך הממויין, בכדי לבדוק אם קיימת התאמה.

(6) קרא את תרגיל 11 בפרק 5. עכשיו תניח שהפריטים מאותו הסוג אינם מקובצים יחד, אלא מעורבים בין שאר הפריטים. בנה טבלה מסודרת של הפריטים והשתמש בשיטת חיפוש בינרי כדי לדעת אם פריט חדש כבר נמצא בטבלה. אם הוא לא נמצא בטבלה יש להוסיף אותו. אם הוא כבר נמצא, יש לחבר את הכמות שנמכרה מנתוני הקלט אל סכום המכירות של אותו פריט באותו חודש. בסיום יש להדפיס את טבלת המכירות המעודכנת.

(7) נתון מערך דו-מימדי TZIYUN[I,K] אשר מכיל את הציון (מ-0 עד 10) של סטודנט I לשאלה K. הווקטור MISHKAL נותן את השקלול של שאלה K בחישוב הציון הסופי.

כתוב תכנית אשר מקבלת כקלט את השמות של N הסטודנטים כשהם ממויינים בסדר אלפביתי, את המערך TZIYUN ואת הווקטור MISHKAL. התכנית צריכה להפיק כפלט רשימה של שמות הסטודנטים והציונים שלהם, כשהיא ממויינת בסדר יורד, לפי הציון.

(8) חברת שיווק גדולה מעסיקה עובדים מכל חלקי הארץ. נתונים שלושה פרטים אודות כל עובד: שם, מספר אישי ושם העיר בה הוא עובד.

כתוב תכנית אשר מפיקה לפי הוראת המפעיל אחד מבין הדוחות הבאים:
א. שם העובד, מספר אישי ועיר. המיון לפי שם העובד.
ב. מספר אישי, שם העובד ועיר. המיון לפי מספר אישי.
ג. שם עובד ומספרו האישי. שם העיר יופיע ככותרת לכל רשימת העובדים באותה עיר. הרשימה ממויינת לפי שמות העובדים בכל עיר.

(9) משרד שליחויות מקבל חבילות למשלוח אם משקלן איננו עולה על 10 ק"ג ואם

כל אחת ממידות האורך, הרוחב והגובה שלהן אינה עולה על חצי מטר. דמי המשלוח נקבעים על פי המשקל:

משקל:	1	2	3	4	5	6	7	8	9	10
מחיר:	1.5	2.0	2.5	3.0	3.4	3.8	4.2	4.5	4.8	5.0

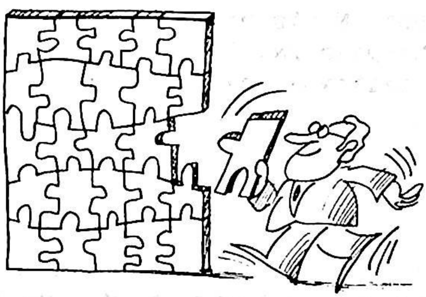
אם מידות החבילה חורגות, אך סכום של שני מימדים איננו עולה על מטר אחד והסכום של שלושת המימדים איננו עולה על מטר וחצי, אפשר לקבל את החבילה למשלוח במחיר כפול, על פי משקלה בקילוגרמים. נתוני הקלט הם:
מספר סידורי של הפעולה, משקל החבילה ושלוש המידות שלה.

הפלט הוא דוח עבודה, אשר כולל נתונים שונים בהתאם לתנאי מסוים:
- אם החבילה קבילה: מספר סידורי של הפעולה ודמי המשלוח שנגבו.
- אם החבילה התקבלה למשלוח במחיר כפול, יש לציין זאת בהערה.
- אם החבילה לא התקבלה יש להדפיס את נתוני הקלט והערה מתאימה.

בסוף הדוח יש להדפיס את מספר החבילות שהתקבלו, את סכום דמי המשלוח שהתקבלו עבורן ואת מספר החבילות שנדחו.

(10) נתונות רשימות ציונים של סטודנטים ב-5 קורסים. בכל רשימה נתונים השם של כל סטודנט שהשתתף בקורס והציון שלו באותו קורס. כל רשימה מסודרת בסדר אלפביתי. סטודנט לא חייב ללמוד בכל הקורסים ולכן שמו לא חייב להופיע בכל רשימה.

יש להדפיס רשימה של שמות כל הסטודנטים לפי סדר אלפביתי. לכל סטודנט יש לרשום את הקורסים שהשתתף בהם ואת הציון הממוצע שלו בקורסים אלו.



פרק 8

פרוצדורות ופונקציות

8.1 מבוא

קטע הוא חלק לוגי של האלגוריתם לפתרון הבעיה, שאפשר לתחום אותו כיחידה שלימה. באלגוריתמים שבהם ישנם חלקים שיש לבצע פעמים אחדות במהלך הפתרון, השימוש בשיטת הקטעים יכול לפשט הן את המבנה הלוגי של הפתרון והן את התכנות. כאשר כותבים תרשים זרימה לתכנית מתארים בתרשים נפרד כל קטע שמוגדר בה. כאשר מקודדים תכנית שהתרשים שלה מחולק לקטעים, אפשר לכתוב את הקטע בנפרד, או לכתוב אותו במקומו בגוף התכנית.

בתכניות שהודגמו עד עכשיו הוצגה רק האפשרות לכתיבת קטעים בגוף התכנית, בדרך כלל בצורה של לולאה. יש לכך מספר חסרונות:

א. התכנית ארוכה ובכך קשה יותר להבדיל בין המהלך העיקרי שלה לבין הפרטים. אפשר לכתוב הערות ליד כל קטע אשר מתחיל במלה BEGIN, אבל אין בכך פתרון לבעיה החשובה של הצגת המהלך כולו. כאשר נוקטים בדרך זו צריך ללמוד את כל התכנית לפני שאפשר לטפל בחלקים שלה.

ב. כאשר יש צורך לבצע קטע תכנות פעמים אחדות כותבים אותו מתחילתו עד סופו בכל מקום שהוא דרוש.

כדי להתגבר על בעיות אלו משתמשים בשיטות תכנות של פרוצדורות (Procedures) ופונקציות (Functions). אפשר לכנות פרוצדורה ופונקציה בשם הכולל שגרה (Subroutine), אשר מתארת תכנית לביצוע פעולה מצומצמת שכיחה, או פעולה אשר חוזרת פעמים אחדות בתכנית העיקרית. בהמשך נלמד מתי ואיך להשתמש בהן.

8.2 פרוצדורות

בתכנית הבאה נכתוב גירסה חדשה של התכנית MASKORET3 שהוצגה בפרק 3. תרשים הזרימה נשאר ללא שינוי.

```
(1)  PROGRAM MASKORET14 (INPUT, OUTPUT);
(2)        (* תכנית זאת מחשבת את המשכורת של כל 50 הפועלים במפעל *)
(3)  VAR
(4)   SHIUR, SHAOT, SCHAR: REAL;
(5)   MONE: INTEGER;

(6)  PROCEDURE CHISHUV;   (* בתרשים שבפרק 3, קטע זה נקרא "עיבוד" *)
(7)    BEGIN
(8)      READ (SHIUR, SHAOT);
(9)      SCHAR:= SHIUR * SHAOT;
(10)     WRITELN (SCHAR:8:2, '^שעות הוא', SHAOT:6:1, '^השכר עבור');
(11)     MONE:= MONE+1
(12)   END;

(13) BEGIN                               (* התכנית הראשית *)
(14)    MONE:=1;
(15)    WHILE MONE <= 50 DO CHISHUV      (* קריאה לפרוצדורה *)
(16) END.
```

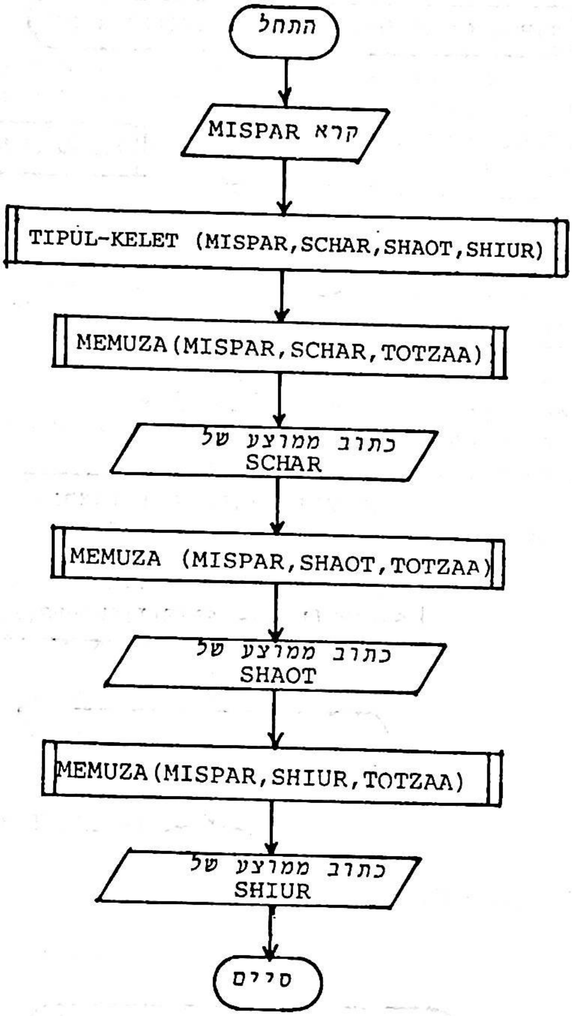
<u>הסבר:</u>

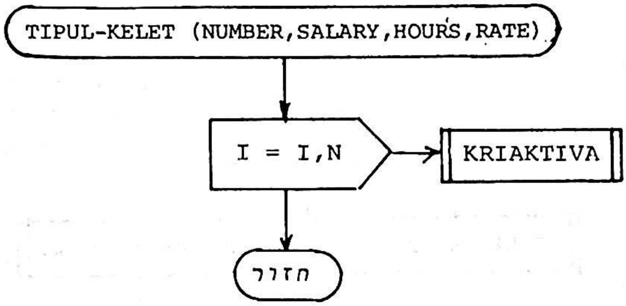
בתכנית זו לא כתבנו את הקטע "CHISHUV" בגוף התכנית כמו שעשינו ב-MASKORET3, אלא רק את המלה CHISHUV. הפירוט של הפקודות בקטע זה נעשה בפרוצדורה בעלת שם זהה, החל משורה (6). בשורה (6) כותבים את המלה PROCEDURE ואחריה את שם הפרוצדורה, אשר משתמשים בו להפעלתה בהמשך התכנית.

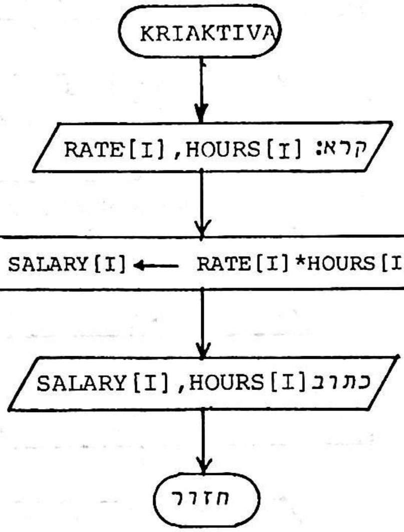
את <u>הצהרת הפרוצדורה</u> (Procedure Declaration) שבשורה (6) כותבים תמיד אחרי הצהרת המשתנים של התכנית. המשתנים שנמצאים בתוך הפרוצדורה הוצהרו בהצהרת המשתנים של התכנית הראשית. כל משתנה שמוצהר בתכנית הראשית וכל ערך המיוחס אליו, תקף גם עבור הפרוצדורה. כל שינוי בערכו של משתנה כתוצאה מביצוע הפרוצדורה, תקף לגבי התכנית כולה.

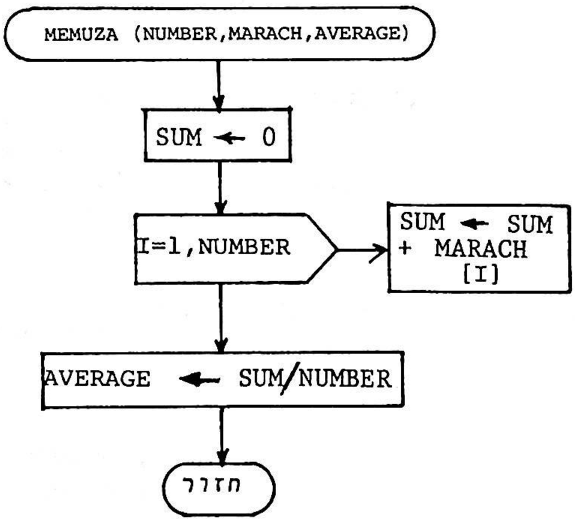
בדוגמה הבאה נשנה את התכנית הקודמת כדי שתוכל לטפל בכל מספר של פועלים, ולא רק ב-50. בנוסף לזה, נחשב את השכר הממוצע, מספר שעות עבודה הממוצע וגם את שיעור השכר הממוצע לשעה של הפועלים. מכיון ש-3 הפעולות האחרונות עוסקות בחישוב ממוצעים, כדאי להגדיר פרוצדורה אחת לחישוב ממוצעים ולקרוא לה 3 פעמים.

תרשים זרימה לתכנית:


התחל
קרא MISPAR
TIPUL-KELET (MISPAR,SCHAR,SHAOT,SHIUR)
MEMUZA(MISPAR,SCHAR,TOTZAA)
כתוב ממוצע של SCHAR
MEMUZA (MISPAR,SHAOT,TOTZAA)
כתוב ממוצע של SHAOT
MEMUZA(MISPAR,SHIUR,TOTZAA)
כתוב ממוצע של SHIUR
סיים


TIPUL-KELET (NUMBER,SALARY,HOURS,RATE)
I = I,N
KRIAKTIVA
חזור


KRIAKTIVA
RATE[I],HOURS[I] :קרא
SALARY[I] ← RATE[I]*HOURS[I]
SALARY[I],HOURS[I]כתוב
חזור


MEMUZA (NUMBER,MARACH,AVERAGE)
SUM ← 0
I=1,NUMBER
SUM ← SUM + MARACH [I]
AVERAGE ← SUM/NUMBER
חזור

הסבר:

התרשים הראשי משתמש בשתי פרוצדורות עיקריות. הראשונה, TIPUL_KELET, מפקחת על לולאה שקוראת לפרוצדורה משנית שקוראת את הנתונים, מחשבת שכר ומדפיסה. הפרוצדורה העיקרית השנייה, MEMUZA, מחשבת ממוצעים של מספרים שנמצאים במערך. משתמשים שלוש פעמים בפרוצדורה לחישוב הממוצע: בכדי לחשב שכר ממוצע, ממוצע השעות ושיעור השכר הממוצע לשעה. לפרוצדורה בתכנית זו קשורים מספר משתנים שיוסברו בהמשך.

כאשר משתמשים בפרוצדורה בתכנית הראשית יש לרשום את שמה (למשל MEMUZA, או TIPUL_KELET) ולאחריו בסוגריים - את השמות של המשתנים הנדרשים בפרוצדורה. בתוך הפרוצדורה עצמה, מכינים רשימה מקבילה של משתנים, שבה נמצא לרוב שמות אחרים, כי הפרוצדורה מיועדת לשימוש כללי ונכתבת לעתים בנפרד מהתכנית הראשית. לכן, במקום MISPAR כתבנו NUMBER בפרוצדורות TIPUL_KELET ו-MEMUZA ובמקום SHIUR, SHAOT ו-SCHAR השתמשנו במערכים הכלליים HOURS, RATE ו-SALARY בהתאמה. בפרוצדורה MEMUZA השתמשנו בשם MARACH שהוחלף בכל קריאה ע"י אחד מן המערכים SHIUR, SHAOT או SCHAR.

התכנית הבאה מבוססת על התרשימים הקודמים:

```
PROGRAM MASKORET15 (INPUT,OUTPUT);
TYPE
  ARRAYTYPE = ARRAY [1..100] OF REAL;
VAR
  SCHAR, SHAOT, SHIUR: ARRAYTYPE;
  MISPAR: INTEGER;
  TOTZAA: REAL;

PROCEDURE TIPUL_KELET
                  (NUMBER:INTEGER;
                   VAR SALARY, HOUR, RATE: ARRAYTYPE);
 VAR
    I:INTEGER;

 PROCEDURE KRIA_KTIVA;
   BEGIN
      WRITELN ('הכנס מס'' שעות ושכר לשעה');
      READLN (HOURS[I], RATE[I]);
      SALARY[I]:= HOURS[I]*RATE[I];
      WRITELN (SALARY[I]:6:2, '^שעות הוא^',
                         HOURS[I]:6:2, 'השכר עבור^')
   END;

  BEGIN                    (* TIPUL_KELET פרוצדורה *)
   FOR I:=1 TO NUMBER DO
     KRIA_KTIVA                 (* קריאה לפרוצדורה *)
  END;

PROCEDURE MEMUZA (NUMBER: INTEGER;
                  MARACH: ARRAYTYPE;
```

```
(27)                 VAR AVERAGE:REAL);
(28)    VAR
(29)        I: INTEGER;
(30)        SUM: REAL;
(31)    BEGIN                              (* MEMUZA פרוצדורה *)
(32)      SUM:=0;
(33)      FOR I:=1 TO NUMBER DO
(34)        SUM:= SUM + MARACH[I];
(35)      AVERAGE:= SUM/NUMBER
(36)    END;

(37)  BEGIN                               (* התכנית הראשית *)
(38)     WRITELN ('כמה עובדים במפעל?');
(39)     READLN (MISPAR);
(40)     TIPUL_KELET (MISPAR, SCHAR, SHAOT, SHIUR);
(41)     MEMUZA (MISPAR, SCHAR, TOTZAA);
(42)     WRITELN (TOTZAA:8:2, 'השכר הממוצע הוא^');
(43)     MEMUZA (MISPAR, SHAOT, TOTZAA);
(44)     WRITELN (TOTZAA:8:2;  ,מספר ממוצע של שעות הוא);
(45)     MEMUZA (MISPAR, SHIUR, TOTZAA);
(46)     WRITELN (TOTZAA:8:2, 'השכר הממוצע לשעה הוא^');
(47)  END.
```

<u>הסבר:</u>

(2)-(7) בשורות אלו מגדירים <u>משתנים גלובליים</u> (Global Variables) ו<u>סוגים גלובליים</u> (Global Types). אלה הם משתנים וסוגים שיהיו מוגדרים במהלך כל התכנית ונוכל להשתמש בהם בכל פרוצדורה בלי כל הצהרה נוספת.

(37)-(47) קטע זה מהווה התכנית הראשית אשר מקבילה לתרשים הראשי. לאחר קליטת מספר העובדים (MISPAR), קוראים לפרוצדורה TIPUL_KELET בשורה (40).

<u>פקודת הפרוצדורה TIPUL_KELET:</u>

שורה (40) נקראת <u>פקודת הפרוצדורה</u> (Procedure Statement). היא כוללת את שם הפרוצדורה ובסוגריים - רשימת <u>פרמטרים אמיתיים</u> (Actual Parameters), אשר מהווים את הארגומנטים של הפרוצדורה. הארגומנטים האלה יחליפו את <u>הפרמטרים הפורמליים</u> (Formal Parameters) שבהצהרת הפרוצדורה ובגוף הפרוצדורה עצמה. בתרגיל זה, המשתנה האמיתי MISPAR יחליף את הפרמטר הפורמלי NUMBER. המשתנה MISPAR צריך להופיע בהצהרת VAR (ראה שורה (6)). סוגו מוגדר כ-INTEGER, ולכן הוא יכול להחליף את NUMBER, שסוגו זהה.

על המתכנת לדאוג לכך שסוגו של פרמטר אמיתי יהיה זהה לסוגו של הפרמטר הפורמלי שהוא מחליף, אבל אין צורך להזכיר את סוג המשתנה האמיתי בפקודת הפרוצדורה עצמה. מן הראוי לציין שסדר הפרמטרים חייב להיות זהה בהצהרת הפרוצדורה ובפקודת הפרוצדורה. ניתן לבדוק זאת ע"י השוואה של הפרמטרים האמיתיים בשורה (40) עם הפרמטרים הפורמליים בשורות (8)-(10).

פקודת פרוצדורה שייכת לחלק הביצועי של התכנית ומצפים לכך שהיא תתבצע כאשר

רושמים את שמה ובסוגריים - את רשימת <u>הפרמטרים האמיתיים</u>. במהלך הביצוע, הפרמטרים הפורמליים יוחלפו ע"י הערכים הנוכחיים של הפרמטרים האמיתיים. במקום MISPAR אפשר היה לכתוב את המספר 10, את הביטוי 10*MISPAR+19, או ביטוי אריתמטי כלשהוא שערכו הוא מספר שלם. בזמן הביצוע, המחשב יעריך את הביטוי ויעביר את ערכו המספרי למשתנה NUMBER וישתמש בו בפרוצדורה.

משתנה כמו NUMBER נקרא <u>פרמטר ערכי</u> (Value Parameter) כי הוא מקבל את הערך של הפרמטר האמיתי, מחסין אותו במשתנה NUMBER ומשתמש בו בפרוצדורה. אין צורך להגדיר בתכנית משתנה אשר יקבל את הערך הסופי של NUMBER, כי ערך זה אינו דרוש בתכנית הראשית.

אם היינו כותבים את המלה VAR לפני NUMBER (כמו שכתבנו VAR בשורה (27) לפני AVERAGE) היינו מודיעים לתכנית לא להגדיר משתנה חדש בשם NUMBER, אלא לייחס שם זה למשתנה MISPAR, שהוא פרמטר אמיתי שקיים בתכנית הראשית. במצב כזה, הפרמטר האמיתי חייב להיות משתנה ולא ביטוי אריתמטי, כדי שנוכל לייחס אליו את השם NUMBER.

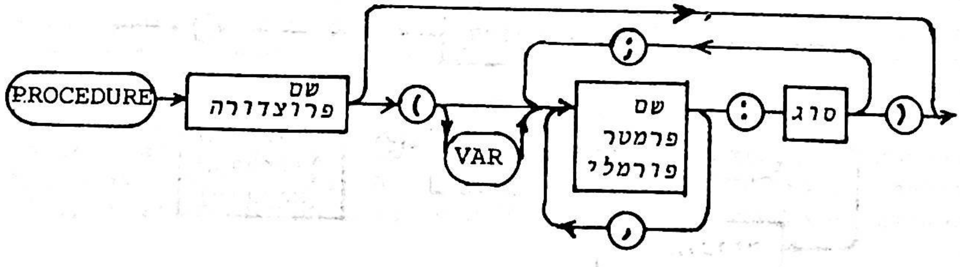
פרמטר כזה נקרא <u>פרמטר משתנה</u> (Variable Parameter), כי מותר להחליף אותו רק בפרמטר אמיתי שהוא <u>משתנה</u> ולא בפרמטר שהוא ביטוי אריתמטי. משתמשים בפרמטר משתנה כאשר רוצים לא רק לקבל נתונים מן <u>התכנית הקוראת</u> (Calling Program), אלא גם להחזיר לה תוצאות כלשהן. המהדר מטפל בהעברת התוצאות אל הפרמטר האמיתי, אשר מחליף את הפרמטר המשתנה.

<u>הצהרת הפרוצדורה TIPUL KELET:</u>

(10)-(8) זוהי <u>הצהרת פרוצדורה</u> מקיפה. אחרי המלה PROCEDURE ושם הפרוצדורה כותבים בין סוגריים את <u>הפרמטרים הפורמליים</u>, דבר שלא ראינו בתכנית MASKORET14. פרמטרים אלה נקראים כך מכיון שהם נכללים בהצהרת הפרוצדורה כדי לייצג את סוגי הארגומנטים שמשתמשים בהם. למשל, NUMBER הוא הפרמטר הראשון וסוגו הוא INTEGER. הפרמטר NUMBER לא הוגדר בהצהרת VAR ולכן הוא קיים רק כאב טיפוס (ז"א בצורה פורמלית) ולא במציאות. כאשר קוראים לפרוצדורה, בשורה (40) מציינים פרמטרים אמיתיים, שיחליפו את הפרמטרים הפורמליים, כפי שיוסבר להלן.

בשורה (10) רשומים 3 פרמטרים משתנים, שמטרתם להחזיר לתכנית הראשית 3 מערכים מלאים שמכילים את השכר, השעות ושיעור השכר לשעה של כל אחד מעובדי המפעל.

הצורה הכללית של הצהרת פרוצדורה היא:



כמו שראינו בתכנית הקודמת, אפשר להגדיר פרוצדורה בלי ארגומנט כלל ולהסתמך על המשתנים שהוגדרו בתכנית הראשית. אפשר גם להגדיר פרמטרים ערכיים ומשתנים לפי הצורך.

בשורה (10) הוגדר הסוג של SALARY, HOURS ו-RATE כ-ARRAYTYPE, אשר מוגדר בתכנית הראשית. קיימים מהדרים שדורשים ששמות סוגים בהצהרת פרוצדורה יהיו מוגבלים למלה אחת ואינם מאפשרים לכתוב "ARRAY [1..100] OF REAL" בתוך ההצהרה. במהדר של VAX למשל, מגבלה זו לא קיימת ואין צורך להגדיר את הסוג הגלובלי ARRAYTYPE.

לאחר הצהרת הפרוצדורה בא גוף הפרוצדורה שנקרא <u>בלוק</u> (Block). הבלוק יכול להכיל את כל ההצהרות המוכרות לנו, דהיינו:

(a) LABEL
(b) CONST
(c) TYPE
(d) PROCEDURE (עם בלוק)
(e) פקודות למיניהן

בפקודה (12) מצהירים על משתנה I. משתנה זה, שמוצהר בתוך הפרוצדורה, נקרא <u>משתנה לוקלי</u> (Local Variable), או <u>משתנה מקומי</u>, כי הוא מוגבל לשימוש של הפרוצדורה בלבד ולא לשימושה של התכנית הראשית. אם נתייחס אליו בתכנית הראשית נקבל הודעה על משתנה שלא הוצהר.

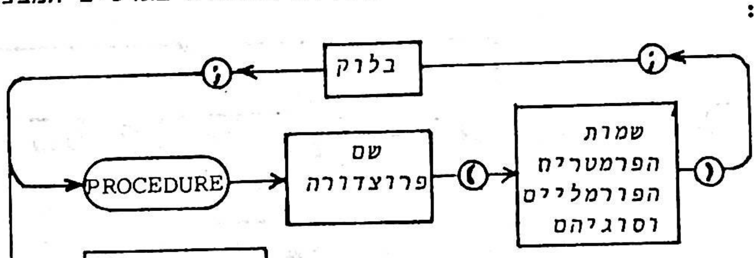
בשורה (13) יש הצהרת פרוצדורה נוספת. אפשר להגדיר פרוצדורה בתוך פרוצדורה בכדי לקבל <u>פרוצדורה מקוננת</u> (Nested Procedure). הפרוצדורה החיצונית יכולה לקרוא לפרוצדורה הפנימית, אך לתכנית הראשית אין גישה אליה. אם היינו רוצים לאפשר גישה, היינו צריכים להצהיר על KRIA_KTIVA כפרוצדורה נפרדת ולרשום אותה לפני ולא בתוך, הפרוצדורה TIPUL_KELET.

הפרוצדורה KRIA_KTIVA ממלאת את המערכים HOURS, RATE ו-SALARY. משתנים אלו הועברו אליה ויועברו חזרה, לא בתור פרמטרים, אלא בתוקף היותם מוגדרים לפרוצדורה TIPUL_KELET שקוראת לפרוצדורה KRIA__KTIVA. אפשר לקרוא להם <u>מערכים גלובליים</u> (Global Arrays) של הפרוצדורה הפנימית KRIA_KTIVA.

(13)-(20) הפרוצדורה הפנימית KRIA_KTIVA.

(21)-(24) החלק הביצועי של הפרוצדורה TIPUL_KELET.

(25)-(36) פרוצדורה נוספת שמקבילה לפרוצדורה TIPUL_KELET (הפרוצדורה KRIA_KTIVA נחותה מהפרוצדורה TIPUL_KELET ומקוננת בה). אפשר להציג את היחס של פרוצדורות מקבילות ופרוצדורות נחותות בתרשים המבנה הכללי של פרוצדורה:



הפרוצדורה KRIA_KTIVA היא חלק של הבלוק של הפרוצדורה TIPUL_KELET ולכן היא נחותה ממנה. הפרוצדורה MEMUZA איננה חלק של הבלוק שכפוף לפרוצדורה כלשהי. היא מייצגת שימוש חוזר של הצהרת פרוצדורה בתוך התכנית הראשית.

כאשר כותבים את תבנית משפט ההצהרות והבלוק של הפרוצדורה, מומלץ לכתוב את המלה PROCEDURE למטה מן המלים PROGRAM, VAR ויתר ההצהרות. את ההצהרות של הפרוצדורה לא רושמים למטה מן המלה PROCEDURE אלא בהזזה ימינה כמו שרואים בשורות (11) ו-(28). אם יש פרוצדורה מקוננת, הצהרת הפרוצדורה הפנימית לא שונה מן ההצהרות האחרות של הפרוצדורה החיצונית והיא מופיעה למטה מהם, כמו שכתוב בשורה (13). שם הפרוצדורה KRIA_KTIVA רשום למטה מהצהרת VAR של הפרוצדורה החיצונית.

8.3 פונקציות

לעתים הקלט לפרוצדורה הוא רשימה של ערכים והפלט הוא ערך אחד שנובע מפעילויות המבוצעות על ערכי הקלט. עבור המקרה הזה קיימת פרוצדורה מסוג מיוחד שקוראים לה <u>פונקציה</u> (Function). השימוש המקובל בפונקציה הוא כאשר:

א. מספר הפרמטרים המשתנים מוגבל ל-1.
ב. הפרמטר המשתנה הוא משתנה רגיל ולא מערך.

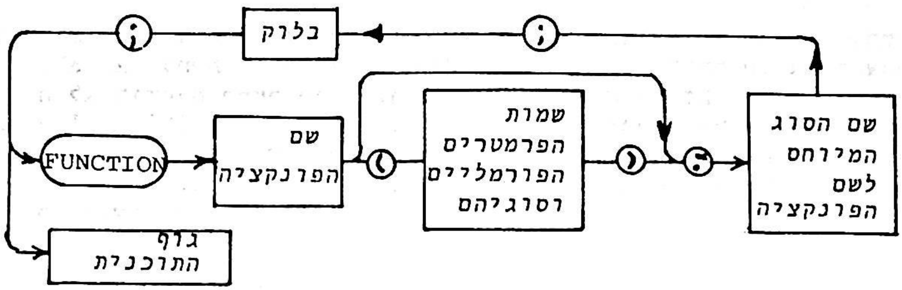
מכאן, שבתכנית הקודמת הפרוצדורה TIPUL_KELET לא יכולה להיות פונקציה כי קשורים אליה 3 פרמטרים משתנים, שהם מערכים. מאידך, עבור הפרוצדורה MEMUZA מוגדר רק פרמטר משתנה פשוט אחד, AVERAGE ולכן היא יכולה להכתב כפונקציה, אשר מוצגת כאן:

```
(1)  FUNCTION MEMUZA (NUMBER : INTEGER;
(2)                   MARACH : ARRAYTYPE) : REAL;
(3)  VAR I : INTEGER;
(4)    SUM : REAL;
(5)  BEGIN
(6)    FOR I:= 1 TO NUMBER DO
(7)        SUM:= SUM + MARACH[I];
(8)    MEMUZA:= SUM / NUMBER
(9)  END;
```

<u>הסבר:</u>

(1)-(2) שורות אלו הן <u>הצהרת הפונקציה</u> (Function Declaration), אשר דומה להצהרת פרוצדורה עם החלפה של המלה PROCEDURE במלה FUNCTION. כותבים תחילה את המלה FUNCTION ואחריה כותבים את השם של הפונקציה: MEMUZA. לכאורה חסרות בשורה (2) המלים AVERAGE:REAL שכתבנו עבור הפרוצדורה MEMUZA, אך אין בזה צורך. מכיון שפונקציה מיועדת רק למקרה שבו התוצאה היא משתנה אחד פשוט, הוסכם שהשם של המשתנה המוחזר יהיה כשם הפונקציה ובמקרה שלנו - MEMUZA. אבל אם MEMUZA מחליף את AVERAGE, איפה הוא התיאור של סוג המשתנה (REAL)? תיאור זה נמצא בסוף שורה (2) אחרי הנקודתיים שלאחר הסוגר הימני של סוף רשימת הפרמטרים הערכיים.

המבנה הכללי של פונקציה:



כדי שהפונקציה תוכל להחזיר ערך, שמה צריך להופיע בצד שמאל של פקודת הצבה, כפי שרשום בשורה (8). שם הפונקציה MEMUZA החליף את המשתנה AVERAGE שהיה הפרמטר המשתנה בפרוצדורה. שם הפונקציה MEMUZA אינו משתנה רגיל ואינו יכול להופיע בצד ימין של פקודת ההצבה בתוך הגדרת הפונקציה. התפקיד היחיד שלו הוא לשמש כתובת לתוצאה שתועבר לתכנית הראשית.

השימוש בפונקציה שונה במקצת משימוש בפרוצדורה. פקודת פרוצדורה, אשר כוללת רשימה של הפרמטרים בפועל, נותנת לנו למעשה פקודה חדשה. לכן היא יכולה לעמוד בפני עצמה, כמו שכתוב בשורות (40), (41), (43) ו-(45) בתכנית MASKORET15. לעומת זאת, פונקציה משתתפת בתכנית בתפקיד של אופרנד ולא של פקודה ולכן היא אינה יכולה לעמוד בפני עצמה. מכאן שאפשר לכתוב שם של פונקציה בצד ימין של פקודת הצבה, עם או בלי פרמטרים נוספים. למשל, בהסתמך על הפונקציה MEMUZA נוכל להחליף את השורה (41) בתכנית MASKORET15 ולכתוב:

```
TOTZAA := MEMUZA (MISPAR, SCHAR);
```

היינו יכולים לכתוב (אם היה צורך בדבר) פקודת הצבה כזו:

```
TOTZAA := MEMUZA (MISPAR, SCHAR) * MEMUZA (MISPAR, SHIUR) + 10;
```

הביטוי בו משתמשים בתוך התכנית לקריאה והפעלה של הפונקציה נקרא <u>מפעיל הפונקציה</u> (Function Designator). הוא מכיל פרמטרים אמיתיים במקום פרמטרים פורמליים, בדיוק כמו בפרוצדורה.

הפונקציה מיועדת למקרה שיש רק פרמטר משתנה אחד, אשר מייחסים אותו לשם הפונקציה. אין הגבלה מבחינת המהדר לכתוב פרמטרים משתנים כפרמטרים פורמליים, אך דרך כתיבה זו נוגדת את הרעיון של השימוש בפונקציה. כשצריך להחזיר יותר מערך אחד, מומלץ להשתמש בפרוצדורה ולא בפונקציה.

יוצא מן הכלל הוא המקרה שהארגומנט הוא מערך, שבו מומלץ להשתמש בפרמטר משתנה בהצהרת הפונקציה. גם במקרה זה רוצים למעשה להשתמש בשם המערך כפרמטר ערכי ולא להחזיר אותו. אבל, אם כותבים את הארגומנט כפרמטר ערכי, הערכים שלו יועתקו למערך מקומי שמוגדר בתוך הפונקציה. לעומת זה, אם כותבים את הארגומנט כפרמטר משתנה, המערך לא יועתק לתוך השטח המקומי של הפונקציה,

אלא רק כתובתו תועבר אליה. העברת הכתובת בלבד יותר יעילה מבחינת הביצוע של התכנית. לכן, במקרה זה, עדיף להעביר את המערך כפרמטר משתנה.

במהדר הסטנדרטי של פסקל יש הגבלה שהמשתנה המיוחס לשם הפונקציה חייב להיות משתנה פשוט ולא מערך, אך יתכן שמהדרים מסוימים יתירו גם משתנה מערך.

כדוגמא נוספת נציג פונקציה בוליאנית, אשר בודקת אם "שנה" שמועברת אליה כפרמטר ערכי היא שנה מעוברת:

```
FUNCTION MEUBERET (YEAR:INTEGER): BOOLEAN;
 BEGIN
   MEUBERET:=(YEAR MOD 4 =0)AND(YEAR MOD 100 <>0)OR(YEAR MOD 400 =0)
 END;
```

התכנית מבליטה את העובדה ששנה תהיה מעוברת אם מספרה מתחלק ב-4, חוץ מסופי מאות. אבל גם שנה שהיא סוף מאה תהיה מעוברת אם מספרה מתחלק ב-400, למשל שנת 2000 הבאה עלינו לטובה.

8.4 שמות לוקליים וגלובליים (*)

שם (Name) הוא כל עצם (Item) שיש להצהיר עליו. בהתאם לכך, העצמים הבאים הם שמות:

א. תוויות (Labels),
ב. קבועים (Constants),
ג. סוגי נתונים (Types),
ד. משתנים (Variables),
ה. פרוצדורות (Procedures),
ו. פונקציות (Functions),
ז. קבועים שהם איברים של סוג המתואר ע"י מנייה (Enumerated Types).

עבור כל שם קיימות:
א. נקודת הגדרה (Point of Definition) בחלק ההצהרות של התכנית.
ב. נקודת יישום (Point of Application), בחלק הביצועי של התכנית.

נקודת ההגדרה של שם חייבת להקדים כל נקודת יישום של אותו שם (חוץ מיוצא מן הכלל אחד שנלמד עליו בהמשך). בדרך זו המהדר יכול לטפל בכל השמות בתכנית במעבר הידור (Compilation) אחד בלבד. התחום (Region) של שם הוא הבלוק שבו הוא מוגדר. ההיקף (Scope) של שם הוא התחום שלו, למעט כל התחומים שיש בהם שם זהה. השם מוכר (Visible) בתוך ההיקף שלו ובמקומות אחרים בתחומו הוא איננו מוכר (Invisible).

אם בלוק A מכיל את בלוק B, נוכל לומר:
א. שם המוצהר ב-A מוכר ב-B אם, ורק אם, לא מצהירים על אותו שם ב-B. שם זה הוא שם גלובלי (Global Name) ב-B.
ב. שם המוצהר ב-A וגם ב-B הוא שם לוקלי (Local Name) ב-B.
ג. אם נקודת יישום של שם המוצהר ב-A נמצאת ב-B, אסור להצהיר על שם דומה ב-B, מכיון שאז הוא מאבד את משמעותו המקורית בתוך בלוק B.

הכללים האלה תקפים לגבי שמות המופיעים בפרוצדורות או פונקציות. בדוגמא שבהמשך נציג את השימוש בכללים האלה:

```
PROGRAM SHEMOT1 (INPUT, OUTPUT);
CONST
     A = 5;
TYPE
     B = (C, D, E);
VAR
    F: B;
    G: INTEGER;

PROCEDURE RISHON;
   CONST
        H = C;
   VAR
       G: B;
       I: INTEGER;
   BEGIN
        התוכן של RISHON
        : :
   END;
PROCEDURE SHENI_CHITZONI;
   TYPE
        A = (H, J, K);
   PROCEDURE SHENI_PNIMI;
      VAR
          B , J:A;
      BEGIN
          התוכן של SHENI_PNIMI
      END;
   BEGIN
       התוכן של SHENI_CHIZONI
   END;
BEGIN
   תוכן התכנית הראשית
   : :
END.
```

הסבר:

(1) את שמות הסוגים המוגדרים אוטומטית, INTEGER ו-REAL, אפשר להגדיר פעם נוספת ועל ידי זה לבטל את הגדרתם המקורית. לא מומלץ לנהוג כך.

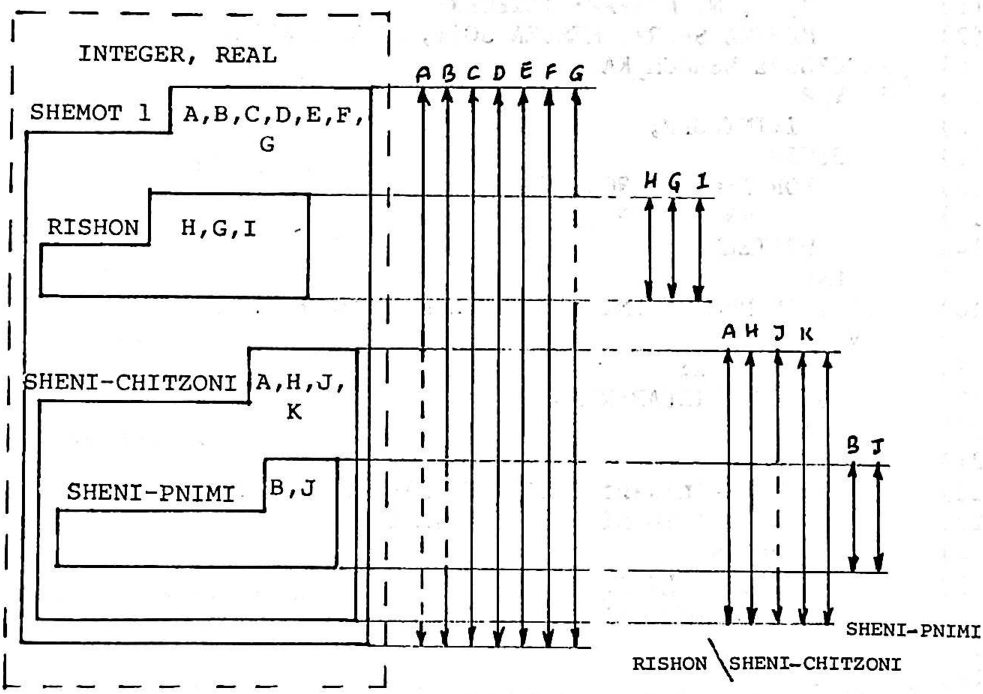
(2) המשתנה השלם G, שהוצהר בתכנית הראשית, לא מוכר ב-RISHON, כי הוא "מוסתר" ע"י משתנה בעל שם זהה שסוגו B. סוג B הוא סוג גלובלי שמוכר גם ב-RISHON.

(3) ב-RISHON משתמשים בקבוע C מסוג B (שהוצהר בתכנית הראשית) כדי להגדיר את הקבוע H. לו היינו מגדירים את B מחדש ב-RISHON, ההגדרה של H הייתה חסרת משמעות.

(4) אין קשר בין המשתנה H המוגדר ב-RISHON וה-H המוגדר ב-SHENI_CHIZONI, כי הם נמצאים בתחומים שונים. המשתנה H ב-RISHON הוא משתנה לוקלי ולכן כתובת הזיכרון שהוא משתמש בה תשוחרר בסיום הביצוע של RISHON.

(5) המשתנה J בפרוצדורה SHENI_PNIMI מסתיר את J של SHENI_CHIZONI. הסוג A שהוצהר בפרוצדורה SHENI_CHIZONI כולל את הערכים H, J, K. H ו-K עדיין תקפים ב-SHENI_PNIMI, אבל J איננו תקף. אפשר להתייחס אל J כמו SUCC(H) או PRED(K), אך עדיף לא להשתמש בסוג A כלל ב-SHENI_PNIMI כדי לא לבלבל את המתכנת ואת הקורא.

התיאור של ההיקף והתחום של השמות השונים בתכנית זו ניתן בתרשים:



בתרשים זה התחום של כל שם מתואר בחץ, כאשר החלק הרציף שלו מייצג את ההיקף של השם והחלק שמקווקו מייצג את החלק בתחומו שבו השם איננו מוכר.

רצוי להגדיר שמות לוקליים. הדבר מונע את האפשרות שערכו של משתנה גלובלי ישתנה כתוצאה מהגדרה של משתנה אחר באותו שם, בין ההצהרה המקורית והשימוש הנוכחי של משתנה זה. כאשר צריכים להחזיק ערך של משתנה בין הפעלה להפעלה של אותה שגרה, אפשר להשתמש בפרמטר משתנה. צריך להגדיר סוג משתנה גלובלי כאשר משתמשים בשם הסוג כדי לתאר סוג של פרמטרים פורמליים בהצהרה של פרוצדורה או פונקציה.

בזמן בדיקת הביצוע של שגרות בונים תכנית ראשית שמטרתה רק לבדוק את השגרה. תכנית זו כוללת את כל הסוגים הגלובליים שמשתתפים בשגרה (וכאמור מוטב למעט בהם) ומנגנון להכניס קלט ולהציג את הפלט בצורה יעילה.

<u>שימוש במשתנים לוקליים:</u>

רוצים לדעת את מידת החום הממוצע בחודש. עורכים מ-3 עד 10 בדיקות כל יום, מוצאים את הממוצע שלהן בכל יום ואת הממוצע של הממוצעים האלה לחודש שלם. נציג את הפתרון בתכנית:

```
PROGRAM GLOBAL_LOCAL (INPUT, OUTPUT);
CONST
     ROCHAV = 10;
     REVACHIM = 5;
VAR
     J, I, N, MISPAR: INTEGER;
     MEMUZA_SHURA, MEMUZA_SOFI, SCHUM: REAL;
PROCEDURE MSHOCH_KAV;
   VAR
      I:INTEGER;
   BEGIN
     FOR I:=1 TO ROCHAV DO
         WRITE ('*');
     WRITELN
   END;
FUNCTION MEMUZA (MISPAR_NETUNIM:INTEGER):REAL;
  VAR
    I:INTEGER;
    SCHUM, MISPAR:REAL;
  BEGIN
    SCHUM:=0;
    WRITELN('הכנס^', MISPAR_NETUNIM, 'נתונים');
    FOR I:= 1 TO MISPAR_NETUNIM DO
       BEGIN
         READ(MISPAR);
         SCHUM:= SCHUM + MISPAR
       END;
    READLN;
    MEMUZA:=SCHUM/MISPAR_NETUNIM
  END;
BEGIN                   (* תכנית ראשית *)
   SCHUM:=0;
   WRITELN ('כמה ימים יכנסו לממוצע הסופי ?');
   READLN (N);
   FOR I:=1 TO N DO
     BEGIN
       WRITELN (I, 'כמה מדידות נעשו ביום^');
       READLN (MISPAR);
       MEMUZA_SHURA:= MEMUZA(MISPAR);
       FOR J:= 1 TO REVACHIM DO
          WRITELN;
```

```
(42)          MSHOCH_KAV;
(43)          WRITELN (MEMUZA_SHURA:ROCHAV:2);
(44)          MSHOCH_KAV;
(45)          SCHUM:= SCHUM + MEMUZA_SHURA
(46)        END;
(47)      WRITELN;
(48)      MSHOCH_KAV;
(49)      MSHOCH_KAV;
(50)      MEMUZA_SOFI:= SCHUM/N;
(51)      WRITELN (MEMUZA_SOFI:ROCHAV:2, '^:ממוצע הממוצעים שווה ל');
(52)      MSHOCH_KAV
(53)    END.
```

<u>הרצה לדוגמא:</u>

```
כמה ימים יכנסו לממוצע הסופי ?
30
^^^^^^^^^1^כמה מדידות נעשו ביום
3
^^^^^^^^^3^הכנס נתונים
4, 6, 7
^^^^^^
^^^^^^
^^^^^^
^^^^^^
^^^^^^

**********
^^^^^^5.67
**********
```

בתכנית זו ישנם משתנים שהוגדרו פעמים אחדות, למשל I, SCHUM, MISPAR. בשורה (40) חייבים להשתמש במשתנה J ולא I, כי הלולאה החיצונית משתמשת כבר ב-I.

כדי להבין את היחס בין המשתנים, נסתכל בציור שבו רשמנו את ההצהרות של המשתנים בתכנית ואת ההגבלות שלהם.

PROGRAM GLOBAL_LOCAL (INPUT, OUTPUT); CONST ROCHAV = 10; REVACHIM = 5; VAR I, N, MISPAR : INTEGER; MEMUZA_SHURA, MEMUZA_SOFI, SCHUM : REAL;	<u>משתנים נראים:</u> GLOBAL_LOCAL, INPUT, OUTPUT ROCHAV, REVACHIM I, N, MISPAR MEMUZA_SHURA, MEMUZA_SOFI, SCHUM MSHOCH_KAV, MEMUZA של התכנית הראשית
PROCEDURE MSHOCH_KAV; VAR I : INTEGER; BEGIN .... END	I של MSHOCH_KAV N, MISPAR, MEMUZA_SHURA, MEMUZA_SOFI, SCHUM של התכנית הראשית.
FUNCTION MEMUZA (MISPAR_NETUNIM : INTEGER) : REAL; VAR I : INTEGER; SCHUM, MISPAR : REAL; BEGIN .... END BEGIN (* תכנית ראשית *) .... END.	MISPAR_NETUNIM, I SCHUM, MISPAR של MEMUZA N, ROCHAV של התכנית הראשית

שים לב לכך שהמשתנה SCHUM שווה למספר חיובי בסוף ביצוע הפונקציה MEMUZA והדבר לא משפיע על ערכו של SCHUM בשורה (45). גם הערך הסופי של I בשורות (23)-(27) לא משפיע על ערכו בשורות (46)-(35), כאשר הוא מציין את מספר הממוצעים שחושבו. גם אין התנגשות בין MISPAR בשורה (38) שמתייחס למספר המדידות שנעשו ביום מסויים ל-MISPAR בשורה (25) שמתייחס לערך של כל מדידה.

השמות של הפרוצדורות הם שמות גלובליים ולכן התחום של השמות MSHOCH_KAV ו-MEMUZA הוא כל התכנית. לכן אסור לשנות את השם של המשתנה MEMUZA_SOFI ל-MEMUZA, מכיון שהשם MEMUZA מוגדר כשם פונקציה באותו תחום (התכנית הראשית).

8.5 מבנים משולבים (*)

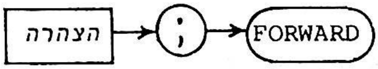
אפשר לחלק את המבנים של פרוצדורה ופונקציה לשני חלקים:

א. <u>הצהרת השגרה</u>, או <u>כותרת</u>.
ההצהרה מורכבת מן האלמנטים הבאים:
- המלה PROCEDURE או FUNCTION.
- שם השגרה.
- שמות הפרמטרים הפורמליים וסוגיהם.
- הסוג של שם הפונקציה, כאשר מגדירים פונקציה.

ב. <u>גוף השגרה</u>, או <u>בלוק</u>.
אחרי ההצהרה כותבים את גוף השגרה, אשר כולל את הפקודות לביצוע.

כאשר מפרידים את גוף השגרה מן ההצהרה, כפי שנראה בהמשך, צריך לכתוב אחריה את מלת ההוראה שנקראת <u>DIRECTIVE</u>.

הצורה הכללית של הצהרה עם DIRECTIVE:



בדוגמא זו, ה-DIRECTIVE היה מסוג FORWARD, שהוא חלק משפת פסקל הסטנדרטית. מלה זו מורה על כך שגוף השגרה יופיע יותר מאוחר בתכנית.

יש שני מקרים בהם רוצים לעשות הפרדה כזאת:

א. כאשר שגרה ברמה מסוימת קוראת לשגרה אחרת, ברמה יותר נמוכה. בלי שימוש ב-FORWARD צריך לכתוב תחילה את השגרה ברמה הנמוכה, כי מתייחסים אליה בהמשך ולאחריה כותבים את השגרה הקוראת. כדי לתקן מצב זה, כותבים רק את הכותרת של השגרה הפחות חשובה ואחריה את המלה FORWARD. את גוף השגרה אפשר לכתוב מאוחר יותר בתכנית.

ב. כאשר שגרה A קוראת ל-B, וגם B קוראת ל-A. במקרה זה חייבים להזכיר את השם של אחת השגרות לפני שנכתוב את גופה. אח"כ כותבים את גוף השגרה השנייה, שתוכל עכשיו לקרוא לראשונה, ששמה כבר הופיע בהצהרה. לבסוף, כותבים את גוף השגרה שהוזכרה ראשונה.

<u>דוגמת תכנות לחישוב מספר ראשוני:</u>
התכנית הבאה מקבלת כקלט מספר J ומפיקה כפלט את המספר הראשוני אשר קרוב ביותר ל-J וגדול ממנו. מספר ראשוני הוא מספר שאינו פריק, ז"א שאינו שווה למכפלה של זוג של מספרים שלמים כלשהו.

```
PROGRAM METZIAT_RISHONI (INPUT, OUTPUT);
TYPE
   MISPAR_CHIYUVI = 1..MAXINT;

PROCEDURE KRIA (VAR MISPAR:INTEGER); FORWARD;
(* פרוצדורה זו קוראת כל מספר שלם, גם אם יש בו פסיק *)

FUNCTION RISHONI  (P:MISPAR_CHIYUVI): BOOLEAN; FORWARD;
(* פרוצדורה זו קובעת אם המספר הוא מספר ראשוני *)
PROCEDURE KRIA_VEHADPASA;
VAR
   K:MISPAR_CHIYUVI;
   J:2..MAXINT;
   NIMTZA:BOOLEAN;
BEGIN
```

```
    WRITELN ('^^^הכנס מספר');
    KRIA(J);
    IF NOT (J>0) THEN
      WRITELN ('מספר שאינו חיובי, אינו קביל בתכנית זו')
    ELSE
       BEGIN
         WRITELN  ('^:הוא', J, '^:המספר הראשוני הקרוב ביותר ל');
         IF RISHONI(J) THEN
            WRITELN (J,  '^המספר')
         ELSE
          BEGIN
            NIMTZA:=FALSE;
            IF ODD(J) THEN
               K:=2
            ELSE
               K:=1;
            REPEAT
               IF RISHONI (J+K) THEN
                  BEGIN
                     NIMTZA:=TRUE;
                     WRITELN (J+K, '^המספר')
                  END;
               K:= K+2
            UNTIL NIMTZA;
            WRITELN
          END
       END
 END;

 FUNCTION  RISHONI;
    VAR
       SHORESH, MECHALEK: MISPAR_CHIYUVI;
    BEGIN
      IF P<4 THEN
        RISHONI:= P>1
      ELSE
        IF NOT ODD(P) THEN
          RISHONI:=FALSE
        ELSE
          BEGIN
            SHORESH:= ROUND (SQRT(P));
            MECHALEK:=3;
            WHILE (MECHALEK<=SHORESH) AND (P MOD MECHALEK<>0) DO
               MECHALEK:= MECHALEK+2;
            RISHONI:= MECHALEK > SHORESH
          END;

 PROCEDURE KRIA;
   VAR
      CH: CHAR;
```

```
(62)      SHLILI: BOOLEAN;
(63)    BEGIN
(64)      MISPAR:=0; (* הערך הסופי של הנתון יהיה במשתנה MISPAR *)
(65)      REPEAT   (* אם יש רווחים בתחילת הקלט, יש לדלג עליהם *)
(66)        READ (CH)
(67)      UNTIL (CH <> ' ') OR EOF;
(68)      IF (CH='+') OR (CH='-') THEN
(69)         BEGIN
(70)           SHLILI:= CH='-';
(71)           READ (CH)              (* קרא את הספרה הראשונה *)
(72)         END
(73)      ELSE                  (* התו הראשון הוא ספרה ולא סימן *)
(74)         SHLILI:= FALSE;
(75)      WHILE (CH <> ' ') AND NOT EOF DO (* קרא תווים והרכב מספר *)
(76)           BEGIN
(77)             IF (CH >= '0') AND (CH <= '9') THEN
(78)               MISPAR:= 10*MISPAR + ORD(CH) - ORD('0');
(79)             READ (CH)
(80)           END;
(81)      IF SHLILI THEN
(82)        MISPAR:= -1*MISPAR
(83)    END;

(84)  BEGIN                  (* תכנית ראשית *)
(85)    WHILE NOT EOF DO
(86)      KRIA_VEHADPASA
(87)  END.
```

<u>הרצה לדוגמא:</u>

הכנס מספר
6,632
המספר הראשוני הקרוב ביותר ל:^6632^^^^^^הוא:^המספר^6637^^^

<u>הסבר:</u>

בכתיבת הפרוצדורה למציאת המספר הראשוני, אשר קרוב למספר J ושווה לו או גדול ממנו, יש לבצע את הפעולות הבאות:

(1) לבדוק אם J הוא מספר ראשוני (שורה 21).

(2) אם J איננו מספר ראשוני, בודקים אם הוא זוגי (שורה 26).

(3) אם הוא זוגי, בודקים את המספר האי-זוגי J+1 שאחריו. אם הוא לא זוגי, בודקים את המספר האי-זוגי J+2 שאחריו (שורה 31).

(4) אם המספר שנבדק בשלב (3) איננו ראשוני, מתקדמים עוד שניים ובודקים לראשוניות (36). ממשיכים בשיטה זו עד שנתקלים במספר ראשוני. בדרך דומה אפשר למצוא את המספר הראשוני הקרוב ל-J וקטן ממנו. לשם כך יש להחליף בשורות (31) ו-(34) את הביטוי J+K בביטוי J-K.

הסבר לפונקציה RISHONI:

כל המספרים, אשר קטנים מ-4 הם ראשוניים, חוץ מ-1. לכן ערכו הבוליאני של P>1 (כאשר P<4) עונה על השאלה אם מספר קטן מ-4 הוא מספר ראשוני (שורה 47). אם התשובה היא TRUE (ז"א המספר הוא 2 או 3), המספר הוא ראשוני. אם P≥4 ואם המספר הוא אי זוגי, צריך לבדוק את כל המספרים מ-1 עד √P ולראות אם אחד מהם מחלק את P בלי שארית. כדי לעשות זאת, מחלקים את P בכל מספר החל מ-3 ועד √P, שורה (55). אם אחד מהמספרים האלה מתחלק ללא שארית, יוצאים מן הלולאה. אפשר לצאת מהלולאה בשני מקרים:

א. כאשר מגיעים עד √P ולא בוצע חילוק ללא שארית.
ב. לא מגיעים עד √P, אבל אחת מפעולות החילוק התבצעה ללא שארית.

כדי לבדוק אם יוצאים מהלולאה מסיבה א' שבה P הוא ראשוני, או מסיבה ב' שבה P איננו ראשוני, מספיק לכתוב את שורה (57).
הסיבה שהלולאה בשורה (55) היתה צריכה להגיע רק עד √P היא שאם P איננו ראשוני, אפשר לכתוב אותו בצורה הבאה: P=A*B.

יש שתי דרכים לפתרון משוואה זו:
א. $A=B=\sqrt{P}$
ב. אחד מן הגורמים A או B קטן מ-√P.

אם P הוא פריק, אחד מן הגורמים חייב להיות קטן או שווה ל-√P. לכן, אם בודקים ולא מוצאים גורם שווה או קטן מ-√P אשר מחלק את P בלי שארית פירושו שאין גורם כזה ו-P הוא מספר ראשוני.

לפני גוף הפונקציה כתוב בשורה (42): FUNCTION RISHONI, בלי פרמטרים. הצהרה זו נקראת <u>זיהוי הפונקציה</u> (Function Identification) שהיא כעין כותרת. היא מתאימה למצב שהתיאור המלא כבר ניתן פעם אחת בתכנית. כאשר ה-DIRECTIVE הוא FORWARD, יש לרשום ליד גוף התכנית רק את זיהוי הפונקציה ללא הצהרה מלאה. הטיפול זהה גם בפרוצדורה KRIA, השווה שורות (4) ו-(59).

הסבר לפרוצדורה KRIA:

פרוצדורה זו עושה דימוי לחלק מן הפעולות שנעשות ע"י הפרוצדורה המוגדרת אוטומטית, READ או READLN. קוראים את תווי המספר כתווים מסוג CHAR, בשורה (66). אם התו הוא סימן מינוס, לוקחים זאת בחשבון (שורה 70). קוראים את הספרות משמאל לימין והופכים כל ספרה מייצוג CHAR לייצוג INTEGER (שורה 78). כופלים כל ספרה ב-10 מספר פעמים בהתאמה למספר הספרות בצד ימין שלה. למשל, ה-6 השמאלי של 6632 יוכפל 3 פעמים ב-10, ה-6 הימני יוכפל פעמיים, ה-3 יוכפל פעם אחת וה-2 לא יוכפל אף פעם כי אין ספרות מימין. אם במספר יש פסיק, תו פיסוק אחר, או אות, הפרוצדורה מתעלמת ממנו וממשיכה הלאה. מכאן שפרוצדורה זו איננה מתאימה לטיפול במספרים ממשיים.

8.5.1 תכנית עם פרוצדורות נפרדות (*)

שתי מלות הוראה, שאינן חלק משפת פסקל הסטנדרטית מיועדות למקרה שגוף

הפרוצדורה לא נמצא בתכנית אלא בנפרד. הראשונה היא FORTRAN ומשתמשים בה כאשר השגרה כתובה בשפת FORTRAN. השנייה היא EXTERNAL ומשתמשים בה כאשר הפרוצדורה כתובה בפסקל, אבל רוצים להדר (to Compile) אותה בנפרד. במצב זה צריכים לכתוב כותרת מלאה גם בהצהרה שנמצאת בתכנית הראשית וגם בהצהרה שנמצאת בפרוצדורה. אם הפרוצדורה משתמשת במשתנים גלובליים, הצהרה של משתנים אלה צריכה להופיע בשגרה.

הדוגמא הבאה מתאימה למהדר מסוים של פסקל. עם זאת אפשר להניח שהעקרונות מתאימים גם למהדרים אחרים.

```
PROGRAM    RASHI (OUTPUT);          (* התכנית הראשית *)
VAR
     X,Y: INTEGER;
PROCEDURE SUB1 (L: INTEGER); EXTERNAL;
PROCEDURE SUB2; EXTERNAL;
PROCEDURE SUB3; EXTERNAL;
BEGIN
   X:=5;
   Y:=8;
   SUB1(X);
   SUB2;
   SUB3
END.                                (* סיום התכנית הראשית *)
```

התוכן של פרוצדורה SUB1:

```
PROCEDURE SUB1 (L: INTEGER);
   BEGIN
      WRITELN (L)
   END;
```

התוכן של הקובץ שמחזיק את הפרוצדורה SUB2:

```
VAR
   X: INTEGER;          (* הצהרת VAR אינה שייכת לפרוצדורה
PROCEDURE SUB2;            אבל היא כן שייכת לקובץ *)
   BEGIN
      WRITELN(X)
   END;
```

התוכן של הקובץ שמחזיק את הפרוצדורה SUB3:

```
VAR
   X, Y: INTEGER;
PROCEDURE SUB3;
   BEGIN
      WRITELN(Y)
   END;
```

הרצה טיפוסית נותנת את התוצאות האלו:

```
5
5
8
```

הסבר:

השגרה SUB1 מדפיסה את ערכו של הפרמטר X, אשר בדוגמא זו ערכו הוא 5. השגרות SUB2 ו-SUB3 משתמשות במשתנים גלובליים. השגרה SUB2 משתמשת רק ב-X ומכיון שהוא הראשון ברשימת המשתנים, אפשר לכתוב X:INTEGER לפני השגרה. השגרה SUB3 משתמשת ב-Y והיא תקבע לו כתובת יחסית בזיכרון כדי שתוכל לפנות אליו. נוכל לדעת את הכתובת היחסית של המשתנה Y רק אם נרשום את כל המשתנים המוצהרים לפניו בתכנית הראשית ולכן ב-SUB3 רושמים גם את X וגם את Y, אף שלא משתמשים ב-X. אפשר היה להשתמש ב-SUB3 בשמות Q ו-R למשל, בהנחה שגם Q ו-R הם משתנים שלמים, מכיון שלא השמות קובעים אלא הגדרת סוג המשתנים ומקומם. במקרה זה R מקביל ל-Y שבתכנית הראשית.

8.6 תרגילים

8.6.1 שאלות

(1) נניח שבתכנית הראשית כתוב:

```
X:=15;
Y:=20;
TARGIL (X,Y);
WRITELN (X,Y);
```

מה יודפס בכל אחד מן המקרים הבאים:

א.

```
PROCEDURE TARGIL (A,B:INTEGER);
   BEGIN
      A:=A+4;
      B:=B-3;
      WRITELN(A,B)
   END;
```

ב.

```
PROCEDURE TARGIL (VAR A:INTEGER; B:INTEGER);
  BEGIN
    A:=A+4;
    B:=B-3;
    WRITELN (A,B)
  END;
```

ג.

```
PROCEDURE TARGIL (VAR A,B:INTEGER);
  BEGIN
     A:=A+4;
     B:=B-3;
     WRITELN (A,B)
  END;
```

(2) מהו התחום וההיקף של כל השמות בתכנית הבאה:

```
PROCEDURE RASHI (INPUT, OUTPUT);
CONST  E=15;
TYPE L= 0..30;
VAR S, A, B: INTEGER;

PROCEDURE RISHON;
   VAR S, B, C: INTEGER;
   :
   END; (*RISHON סוף*)

PROCEDURE SHENI;
  TYPE P= (B, S, E);
  VAR A, C, D, L: INTEGER;
  :

  PROCEDURE SHLISHI;
    VAR A, E: INTEGER;
    :
    END; (*SHLISHI*)
END;    (* SHENI *)
END;     (* RASHI *)
```

(3) מהו הפלט של התכנית הבאה? יתכן שיהיו בה משתנים בלתי מוגדרים.

```
PROGRAM DUGMA (OUTPUT);
VAR
    A, B, C: INTEGER;

PROCEDURE RISHON (VAR X: INTEGER; Y: INTEGER);
    VAR B, C, D: CHAR;

    PROCEDURE PNIMI;
       VAR  X, Y: INTEGER;
        BEGIN
          WRITELN (A, B, C, D, X, Y);
          A:= ORD(B)
        END;

  BEGIN                                  (* RISHON פרוצדורה *)
     D= 'A';
     WRITELN (A, B, C, D, X, Y);
     X:= Y;
     Y:= 0;
     B:= CHR(A);
     C:= CHR(X);
     PNIMI;
     WRITELN (A, B, C, D, X, Y)
  END;
```

```
BEGIN                                                  (* תכנית ראשית *)
     A:=10;
     B:=20;
     C:=30;
     RISHON(A,B);
     RISHON(D,C);
     RISHON(C,A)
END.
```

8.6.2 תכניות

(1) שיעור המס הוא 20% על הכנסה של זוג נשוי אשר מגיעה עד 1,000 שקל ו-40% מס על הכנסה גבוהה יותר. השיעור הוא 10% על הכנסה של אדם בודד עד 500 ו-60% למעלה מזה. כתוב פרוצדורה עבור כל אחד משני המקרים האלה והשתמש בה בתכנית שקולטת שני משתנים:

- מצב משפחתי: N (נשוי), B (בודד).
- רמת הכנסה.

עליך לחשב את המיסים על פי נתונים אלה.

(2) כתוב תכנית שמקבלת מספר ומדפיסה אותו. לידו היא משרטטת קו המורכב מן הסימן מקף (-), שאורכו יחסי למספר הספרות במספר (זוהי היסטוגרמה). את שרטוט ההיסטוגרמה יש לעשות ע"י פרוצדורה עם משתנה ערכי אחד, אשר שווה למספר הספרות במספר.

(3) כתוב פונקציה שמחשבת את הרווח ממכירת מוצרים. נתונה כמות הפריט שנמכרה (C), עלות קבועה לניהול העסק (K) ועלות שיווק משתנה (M) המיוחסת לכל פריט שנמכר. השתמש בפונקציה זו בתכנית שמדפיסה את מצב הרווח בימים שונים, כאשר הקלט הוא מספר היחידות שנמכרו באותו יום. יש להדפיס כותרת לדוח.

(4) כתוב את הפונקציה SCHUM(SUG,NETUNIM,N) לחישוב סכום הריבועים של האיברים בשורה N או בעמודה N של המטריצה NETUNIM. אם SUG='S' מחשבים את סכום האיברים בשורה, אם SUG='A' מחשבים את סכום האיברים בעמודה. השתמש בפונקציה SCHUM בתכנית שקוראת מטריצה של שלוש שורות ו-4 עמודות ומחשבת את סכום הריבועים של כל שורה וכל עמודה. בפלט יש להציג את המטריצה המקורית ובסוף כל שורה - את סכום הריבועים באותה שורה. למטה מכל עמודה יש לרשום את סכום הריבועים באותה עמודה.

(5) כתוב פרוצדורה שקוראת מערך של שמות וסופרת את מספר השמות בקלט. כתוב פרוצדורה נוספת כדי למיין את השמות. השתמש בפרוצדורות אלו כדי להדפיס רשימה מסודרת של השמות המשמשים כקלט לתכנית.

(6) כתוב פרוצדורה שמקבלת כקלט מחרוזת תווים ושני מספרים N_1 ו-N_2. היא מפיקה את התת-מחרוזת שמתחילה ממקום N_1 במחרוזת המקורית ואורכה N_2 תווים.

(7) כתוב פונקציה שמקבלת כקלט את X ו-Y ומחשבת X בחזקת Y.

(8) כתוב תכנית שמקבלת כקלט שמות וציונים של תלמידים בכיתה לא גדולה

מ-200, מסדרת אותם בסדר יורד בעזרת פרוצדורה למיון ומחשבת את הממוצע ושונות המדגם בעזרת פונקציה. הנוסחא לחישוב הממוצע:

$$M = (X_1 + X_2 + X_3 + X_4 + ... + X_N) / N$$

הנוסחא עבור שונות המדגם היא:

$$S^2 = \frac{N\,(X_1^2 + X_2^2 + X_3^2 + X_4^2 + ... + X_N^2) - (X_1 + X_2 + X_3 + ... + X_N)^2}{N(N-1)}$$

הפלט יכיל את הנתונים הבאים ויוצג כרשימה, שבה שורה לכל סטודנט:

- שם הסטודנט.
- הציון.
- מספר הנקודות מעל (+) או מתחת (-) הממוצע.

בסוף הרשימה יודפס ממוצע הציונים בכיתה וסטיית התקן שלהם (סטיית התקן היא השורש הריבועי של שונות המדגם).

(9) בתרגיל זה ארבעה חלקים:

א. כתוב את הפרוצדורה OTAKIM(C,N) שמדפיסה N פעמים את התו C.

ב. השתמש בפרוצדורה זו בתכנית שמדפיסה משולש שווה שוקיים. בתכנית יש שלושה משתנים:
- גובה המשולש.
- התו שממנו יורכב המשולש.
- מספר הרווחים לפני הקודקוד העליון.

אם שלישית המשתנים נראית כך, (10,'*',3), נקבל את התדפיס הבא:

```
          *      הקודקוד יהיה 10 רווחים מן השפה השמאלית של
         ***     המסך
        *****
```

ג. כתוב פונקציה TAV_ACHRON שיש לה פרמטר משתנה אחד, MARACH_80, אשר מוגדר: PACKED ARRAY[1..80] OF CHAR. הפונקציה מחזירה את המספר הסידורי של התו האחרון במערך, שאיננו תו רווח.

ד. כתוב תכנית שמשתמשת בפרוצדורות OTAKIM ו-TAV__ACHRON כדי לקרוא מחרוזת, שמיועדת לכותרת. התכנית מדפיסה את התווים עד התו האחרון במחרוזת ששונה מרווח וממרכזת את מחרוזת הפלט בדף, או במסוף של 80 עמודות, כאשר יש 10 כוכביות משני צידיה.

(10) בתרגיל זה מספר חלקים:

א. כתוב פונקציה ORECH שמקבלת מספר שלם ומפיקה את מספר הספרות במספר.

ב. כתוב פונקציה KRA_AD_PSIK שקוראת שורת קלט עד הפסיק הראשון, כותבת את מחרוזת התווים (בלי הפסיק) למערך באורך 80 וסופרת את מספר התוים עד הפסיק. אם מקבלים EOLN לפני שמגיעים לפסיק, מדפיסים הודעת שגיאה ומדלגים לסוף הפרוצדורה ע"י פקודת GOTO (שמותרת במקרה חריג זה).

ג. שורות הקלט מכילות שני ערכים NAME ו-NUMBER, המופרדים זה מזה ע"י פסיק. כתוב תכנית שתדפיס רשימה של השמות והמספרים לפי המבנה הבא:

```
מספר עמודה:        10        28
                   ¦---- 19 תווים ---¦
תוכן השורות:       YAKOV...........279
                   AVRAHAM..........42
                   GAVRIELA..........1
```

השם מתחיל בעמודה 10 והספרה הימנית של המספר תהיה בעמודה 28. השתמש בפרוצדורה שכתבת בחלקים א' ו-ב' וגם בפרוצדורה OTAKIM מן התרגיל הקודם.

(11) (*) כתוב תכנית שקוראת מחרוזת, מחפשת את כל המופעים של צירוף מסוים (PATTERN) ומחליפה את הצירוף הזה במחרוזת חליפית. המחרוזות הן מן הסוג MACHROZET:ARRAY[1..100]. כל מחרוזת מסתיימת ב-"/", כאשר ה-"/" אינו נחשב כחלק ממנה. אפשר להעזר בפרוצדורות ובפונקציות אלו:

(a) PROCEDURE KRA_AD_GVUL (VAR M: MACHROZET; VAR ORECH: INTEGER)

פרוצדורה זו קוראת מן המסוף מחרוזת M וקובעת את אורכה עד התו "/".

(b) FUNCTION TEUM (M1, M2: MACHROZET; N1, N2, N3: INTEGER): BOOLEAN;

הפונקציה משווה N1 תווים החל מתו N2 במחרוזת M1, אל N3 התווים הראשונים במחרוזת M2.

(c) PROCEDURE MECHIKA (VAR M: STRING; N1, N2: INTEGER)

הפרוצדורה מוחקת N2 תווים במחרוזת M, החל ממקום N1.

(d) PROCEDURE HOSAFE (VAR M1, M2: MACHROZET; N1, N2: INTEGER)

פרוצדורה זו מוסיפה את N2 התווים הראשונים של מחרוזת M2 לתוך מחרוזת M1, החל ממקום N1 ואילך. שאר התווים של M1, מתו N1+1 עד סוף המחרוזת, יהיו אחרי N2 התווים שנוספו ממחרוזת M2.

(12) בתרגיל מספר חלקים:

א. כתוב פרוצדורה בכדי לקרוא מספר אוקטלי ולהפוך אותו למספר עשרוני.

ב. כתוב פרוצדורה נוספת בכדי להפוך מספר עשרוני למספר אוקטלי.

ג. השתמש בשתי הפרוצדורות בכדי לכתוב תכנית שמקבלת סדרה של מספרים אוקטליים, הופכת כל אחד מהם למספר עשרוני, מחברת אותם כמספרים עשרוניים, הופכת תוצאה זו שוב פעם למספר אוקטלי ומדפיסה את התוצאה.

(13) כתוב פרוצדורה שמקבלת שני מספרים אוקטליים ומחברת אותם מבלי להפוך אותם למספרים עשרוניים. השתמש בפרוצדורה זו בתכנית שמחברת 3 מספרים

אוקטליים. את התשובה יש להציג כמספר אוקטלי וגם כמספר עשרוני ע"י שימוש בפרוצדורה שנכתבה בשאלה הקודמת.

(14) הנוסחא לטור טיילור עבור e^X היא:

$$e^X=\Sigma(X^i/i!)=X+(X^2/2!)+(X^3/3!)+...$$

כתוב פונקציה שמחשבת את e^X עד לדיוק של 0.0001, ז"א שההפרש בין האומדן החדש והאומדן הקודם הוא פחות מ-0.0001 כפול האומדן הישן. השתמש בפונקציה זו בתכנית שמחשבת את הערך של e^X עבור X שמשתנה בתחום שבין 0 ל-1 בדילוג של 0.01.

(15) בתרגיל זה שני חלקים:

א. כתוב פונקציה שמנתחת תוצאות מבחנים.
נתוני הקלט:
- N - מספר השאלות במבחן.
- תשובות של כל סטודנט (מערך). התשובות הן מספרים מ-1 עד 4.
- מפתח התשובות (מערך), שהוא אוסף התשובות הנכונות.

הפלט של הפונקציה היא הציון של הסטודנט בבחינה, אשר מחושב לפי מספר התשובות הנכונות מחולק במספר השאלות.

ב. כתוב תכנית שמקבלת את נתוני הקלט הבאים:
- מספר השאלות ומפתח התשובות (מערך),
- מספרי זיהוי של סטודנטים ותשובות של כל סטודנט לכל השאלות.

התכנית מדפיסה את מפתח המבחן בשורה העליונה של הפלט ומתחת לה - את מספרי הזיהוי של הסטודנטים, התשובות של כל סטודנט והציון שלו.

(16) כתוב תכנית בכדי לחשב את ערכו של הביטוי שנותן את מספר הקבוצות בגודל X שאפשר להרכיב מ-N עצמים שונים.

$$\binom{n}{x} = \frac{n!}{x!(n-x)!}$$

הגדר פוקנציה FACT לחישוב עצרת של מספר כלשהו. בתכנית הראשית דרושה רק פקודת הצבה אחת:

```
TOTZAA:= FACT(N) / (FACT(X) * FACT(N-X))
```

(17) (*) נתונים מחירים של מוצרים שונים בדולרים, בערכים אשר קטנים מ-1000$. כתוב תכנית אשר מסבה מחירים אלו לייצוג במלים באנגלית. השתמש בפונקציה שתבצע את הפעולות הבאות:
- תמיר מספר בן שלוש ספרות או פחות למלים.
- בתוכה תהיה פונקציה שמתרגמת ספרה למלה מקבילה, כדי שתשמש להמרה של הספרות השונות של המספר. נשתמש בפונקציה זו גם בעבור ספרת המאות (למשל, FIVE HUNDRED) וגם בעבור ספרת היחידות (למשל, FIVE).

אם, לדוגמא, הקלט הוא 534.18$, הפלט יהיה:
FIVE HUNDRED THIRTY FOUR DOLLARS EIGHTEEN CENTS

(18) כתוב פוקנציה שבודקת אם ספרה מסוימת כלולה במספר. השתמש בפונקציה זו בתכנית שמגלה איזה מספרים I בין 1 ל-100 הם כך שהספרה D (בין 0 ל-9) נכללת ב-I, ב-I^2 וגם ב-I^3.

פרק 9

יישומים של פרוצדורות ופונקציות

9.1 מספרים אקראיים

שגרה הכתובה לעתים קרובות כפרוצדורה, או פונקציה חיצונית היא השגרה ליצירת מספרים אקראיים. מספרים אקראיים יוצרים באחת מתוך שלוש שיטות:

א. שיטת "סיבוב" של גלגל רולטה. על יסוד שיטה זו מכנים את כל השיטות המשתמשות במספרים אקראיים בשם שיטות מונטה קרלו (Monte Carlo Methods).

ב. שיטת אמצע הריבוע (Mid-Square Method), אשר הוצעה ע"י פון-ניומן, ממציא שיטת התכנית המוחסנת בזיכרון.

ג. השיטה הקונגרואנטית (Congruential Method).

נסקור בקצרה את שתי השיטות האחרונות, בעוד שלשיטות מונטה-קרלו נקדיש סעיף נפרד.

9.1.1 שיטת אמצע הריבוע

לחישוב מספר אקראי בשיטה זו נבחר במספר כלשהו בן N ספרות (מספר הספרות N הוא זוגי), אשר נקרא לו גרעין (Seed) ונסמן אותו ב-X_0. נעלה את X_0 בריבוע ונקבל X_0^2. נחלץ את הספרות האמצעיות שלו כדי לקבל את X_1. נחזור על פעולה זו עם X_1 כדי לקבל את X_2 וכו', עד שנקבל כמה מספרים אקראיים שצריכים.

לדוגמא, נבחר מספר בן 4 ספרות ונחלץ תמיד 4 ספרות אמצעיות מתוצאת חישוב הריבוע:

$$X_0 = 4556$$

$$X_0^2 = 11\underline{9439}36$$

אם נמשיך בדרך זו, נקבל את סדרת המספרים הבאה:

$X_1=9439$	$X_7=2568$	$X_{13}=7081$	$X_{19}=0166$
$X_2=0947$	$X_8=5946$	$X_{14}=1405$	$X_{20}=0275$
$X_3=8968$	$X_9=3549$	$X_{15}=9740$	$X_{21}=0756$
$X_4=4250$	$X_{10}=5954$	$X_{16}=8676$	$X_{22}=5715$
$X_5=0625$	$X_{11}=4591$	$X_{17}=2729$	$X_{23}=6612$
$X_6=3906$	$X_{12}=2590$	$X_{18}=4474$	$X_{24}=7185$

חסרון השיטה: אם לא נבחר את X_0 בצורה נכונה, נוכל לקבל לולאה קצרת טווח.

לדוגמא: $X_0 = 7492$

$$X_0^2 = 55\underline{1300}64$$

כאשר ממשיכים לחשב מקבלים:

$X_1=1300$	$X_5=4100$	$X_9=4100$
$X_2=6900$	$X_6=8100$	$X_{10}=8100$
$X_3=6100$*	$X_7=6100$*	$X_{11}=6100$*
$X_4=2100$	$X_8=2100$	$X_{12}=2100$

בדוגמא זו נוצרה לולאה קיצרת טווח שיש בה מספרים שחוזרים על עצמם. המספר החוזר 6100 סומן בכוכבית.

9.1.2 השיטה הקונגרואנטית

אם X_0 הוא הגרעין של המספר, אזי אפשר לקבל סדרה של ערכים בעזרת הנוסחא:

$$X_{i+1} = (AX_i+B) \text{ MOD } M$$

כאשר: A הוא המכפיל,
B הוא הקבוע לחיבור (המחובר),
M הוא המודולוס.

- אם B שווה ל-0, לפנינו שיטה קונגרואנטית כפלית.
- אם A שווה ל-1, לפנינו שיטה קונדרואנטית חיבורית.

הערך של המודולוס M הוא בדרך כלל חזקה של בסיס המספרים שמשתמשים בו. אם M הוא בסיס המספרים, החישוב של "MOD M" פשוט למדי: הוא מורכב מהספרות הימניות של תוצאת הביטוי AX_i+B.

לדוגמא נתון: $M=10$ $A=6$ $B=1$ $X_0=1$

נחשב ונקבל:

$$X_1= (AX_0+B) \text{ MOD } M = (6*1+1) \text{ MOD } 10 = 7$$
$$X_2= (AX_1+B) \text{ MOD } M = (6*7+1) \text{ MOD } 10 = 3$$
$$X_3= (AX_2+B) \text{ MOD } M = (6*3+1) \text{ MOD } 10 = 9$$
$$X_4= (AX_3+B) \text{ MOD } M = (6*9+1) \text{ MOD } 10 = 5$$

גם בשיטה זו יש אפשרות להכנס ללולאה קצרת טווח. מתמטיקאים הקדישו זמן רב לבחירת ערכים מתאימים של הפרמטרים בשיטות ב' ו-ג' כדי שהמחזור יהיה ארוך ככל היותר.

כאשר מחשבים מספרים אקראיים בעזרת מחשב, הערך המתאים ביותר עבור M שווה לגודל מלת המחשב. נוכל להשתמש בנוסחא $X_{i+1}=AX_i+B$, בלי לחלק ב-M. בזמן

ההכפלה יגלוש החלק של המכפלה אשר גדול מן המספר הגדול ביותר שהמחשב יכול להחזיק ונקבל שארית. למשל, מחשב עם מלה בת 16 סיביות יכול להחזיק מספר חיובי שערכו עד $2^{15}-1$, דהיינו 32,767 והערך של M יוכל להיות 32,768. בכל פעם שמקבלים מספר יותר גדול, כתוצאה מכפל למשל, הספרות המשמעותיות תגלושנה ובמלת המחשב נקבל את השארית. אם התוצאה שלילית, המספר שנשאר באוגר יתורגם כמשלים ל-2 של השארית. נוכל להפוך אותה למספר חיובי ע"י כך שנחבר לו 32,767 ועוד 1, כלומר ע"י מחרוזת סיביות של 1 עם 15 אפסים מימין. ה-1 יהפוך ל-0 את ה-1, שכבר נמצא בסיבית הסימן ושגורם למספר להיראות כמספר שלילי והמספר יחשב שוב כמספר חיובי.

כל זה נאמר בהנחה שכתוצאה מגלישה הנובעת מכפל של מספרים שלמים, המחשב לא מפסיק את פעילותו וגם לא מתריע. אם זה לא המצב, שיטה זו איננה מתאימה.

שיטה קונגרואנטית טובה דורשת אם כן, שני דברים:
א. הקפדה בבחירת הפרמטרים.
ב. הכרה טובה של מערכת ההפעלה של המחשב.

אם לא נרצה להתבסס על התכונות המיוחדות של המחשב שבו תופעל התכנית, נוכל לבחור מודולוס עשרוני ולעשות את החילוק ממש, בלי להסתמך על שיטת הטיפול בגלישות.

יש כמה דרישות כדי להבטיח מחזור באורך מקסימלי, כאשר משתמשים בנוסחא הבאה

$$X_{i+1} = (AX_i + B) \text{ MOD } M$$

א. $M > X_0$, A, B,
ב. אין ל-A ו-B אף גורם משותף,
ג. כל גורם ראשוני של M הוא גם גורם ראשוני של (A-1),
ד. אם M הוא כפולה של 4, גם (A-1) הוא כפולה של 4.

דוגמא של פונקציה לחישוב של מספרים אקראיים:

```
FUNCTION RANDOM ( X: INTEGER ) : INTEGER;
CONST
    A = 5;
    B = 32719;
    M = 65437;
BEGIN
    RANDOM:= (A*X+B) MOD M
END;
```

כאשר משתמשים בשיטה זו, צריכים לדאוג לכך שהפלט (RANDOM) יכנס בפעם הבאה כקלט למחשב.

דוגמא ליישום של חישוב מספרים אקראיים לפי הפונקציה RANDOM:
יש ליצור ולהדפיס 50 מספרים אקראיים בין 0 ל-1.

```
N:= 1;
FOR I:= 1 TO 50 DO
   BEGIN
      N:= RANDOM(N);
      SHEVER:= N/65436;
      WRITELN (SHEVER)
   END;
```

מחלקים במספר 65436 שהוא המספר האקראי המקסימלי שנוכל ליצור כאשר M=65437. שיטה זו מתאימה למחשב עם מלה של 32 סיביות, אשר יכול תמיד להחזיק את A*X+B ללא גלישה. לו כתבנו VAR X בהצהרת הפונקציה, יכולנו להחליף את שתי השורות אחרי המלה BEGIN, בפקודה אחת:

```
SHEVER := RANDOM(N)/65436;
```

דוגמא נוספת לחישוב של מספרים אקראיים:

```
(1)  FUNCTION RANDOM: REAL;
(2)   CONST
(3)           A = 309.0;
(4)           B = 0.203125;
(5)   BEGIN
(6)       RANDOM_NUMBER:= A * RANDOM_NUMBER + B;
(7)       RANDOM_NUMBER:= RANDOM_NUMBER - TRUNC(RANDOM_NUMBER);
(8)       RANDOM:= RANDOM_NUMBER
(9)   END;
```

החיסור בשורה (7) הוא כמו חילוק ב-1000, כי הוא גורם לכך שהמשתנה RANDOM_NUMBER, שהיה מספר מ-0.203125 עד 309.203125, יהיה עכשיו מספר בין 0 ל-1. כאשר משתמשים בשיטה זו, RANDOM_NUMBER חייב להיות משתנה גלובלי, כדי שאפשר יהיה להתייחס לערכו בתוך הפונקציה. אם זוהי פונקציה חיצונית, צריך לקיים את דרישות המהדר לשימוש במשתנה גלובלי בתוך פרוצדורה חיצונית. בתכנית הראשית צריך לתת ל-RANDOM_NUMBER ערך התחלתי. מכאן ואילך, בכל פעם שדרוש מספר אקראי, אפשר לרשום את שם הפונקציה RANDOM בלבד בכדי לקבל מספר אקראי בין 0 ל-1.

<u>חישוב PI (π) בעזרת מספרים אקראיים:</u>

התכנית תציין כמה צעדי חישוב נדרשו לקבלת הפתרון.

```
PROGRAM PI (INPUT, OUTPUT);
CONST
   MISPAR=MAXINT;
VAR
   RANDOM_NUMBER, X, Y, PI: REAL;
   I, BETOCH, AL: INTEGER;
FUNCTION RANDOM: REAL; EXTERNAL;

BEGIN
    RANDOM_NUMBER:= 0.1;
    FOR I:= 1 TO MISPAR DO
       BEGIN
          X:= RANDOM;
          Y:= RANDOM;
          IF X*X + Y*Y < 1.0 THEN
             BETOCH:= BETOCH+1
          ELSE
             IF X*X + Y*Y = 1.0 THEN
                AL:= AL+1
       END;
   PI:= 4.0 * (BETOCH + AL / 2.0) / MISPAR;
   WRITELN ('אומדן של PI ב-', MISPAR, 'צעדים הוא^', PI:13:10)
END.
```

<u>הסבר:</u>

היות והשטח של עיגול ברדיוס של 1 הוא $\pi=\pi R^2$, השטח של הרבע הראשון של אותו עיגול הוא $\pi/4$. אם נבחר נקודות מתוך הריבוע $0\leq X\leq 1$, $0\leq Y\leq 1$, נקבל אומדן של $\pi/4$. נבדוק כמה מן הנקודות תהיינה בתוך העיגול וכמה מהן - מחוץ לו. מכיון שאפשר שחלק מהנקודות תהיינה ממש על הגבול (לפחות במידת הדיוק של המחשב), מייחסים 1/2 מאלו לעיגול עצמו וחצי לשטח שמחוץ לו (בגבולות הריבוע 1*1). כופלים ב-4 את היחס בין מספר הנקודות בתוך העיגול לבין מספר הנקודות שיוצרים כדי לעבור מאומדן של $\pi/4$ לאומדן π.

9.2 שיטות מונטה קרלו

שיטות מונטה קרלו הן שיטות בסטטיסטיקה שמאפשרות לחקות, או לדמות תהליך הסתברותי. <u>החקייה</u>, או <u>הדמייה</u> (Simulation) זו נעשית בארבעה צעדים:

א. קביעת פונקצית ההסתברות $f(x)=P[X=x]$, כאשר $f(x)$ מציינת את ההסתברות שמשתנה מקרי X יקבל ערך כלשהוא x.

ב. חישוב פונקצית ההתפלגות, ז"א פונקצית ההסתברות המצטברת:

$$y = F(x) = P[X\leq x] = \sum_{t=\infty}^{x} F(t)$$

הפונקציה הזו מחשבת את ההסתברות שהמשתנה האקראי יהיה שווה ל-x, או לכל ערך שהוא קטן מ-x.

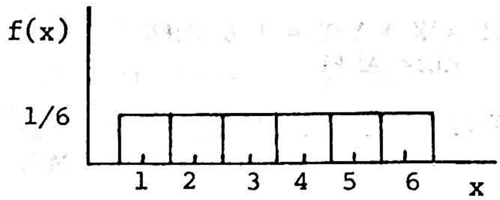
ג. בחירה של מספר אקראי M, אשר נציב אותו כערך של y.

ד. חישוב הערך של x, המיוחס ל-y ע"י הפונקציה: $y = F(x) = P[X \leq x]$.

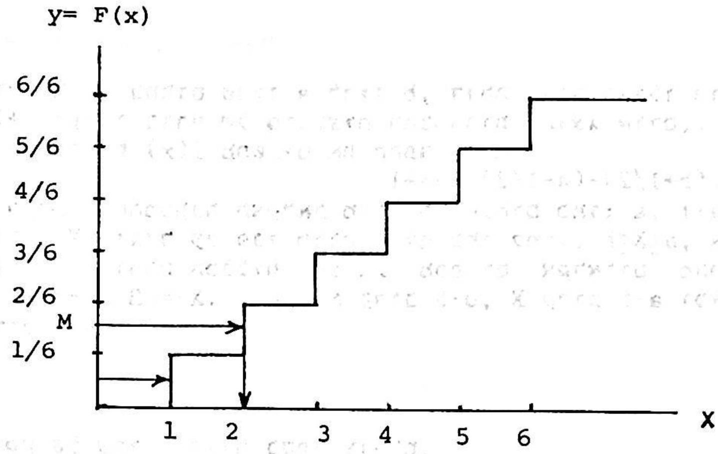
<u>החקייה של זריקת קוביה אחת:</u> פונקצית ההסתברות ופונקצית ההתפלגות רשומות בטבלה שלהלן:

x	f(x)	F(x)
1	1/6	1/6
2	1/6	2/6
3	1/6	3/6
4	1/6	4/6
5	1/6	5/6
6	1/6	6/6

הגרף של פונקצית ההסתברות f(x) הוא:



הגרף של פונקצית ההתפלגות y=F(x) הוא:



הקפיצה בגרף של פונקצית ההתפלגות, כאשר X שווה ל-1, שווה בדיוק להסתברות שהתוצאה של זריקת הקוביה תהיה 1 (ז"א הקפיצה היא מ-0 ל-1/6). כדומה לגבי הקפיצות כאשר X שווה ל-2, 3, 4, 5, ו-6. פירושו של הצעד הרביעי בשיטות מונטה קרלו הוא, שאם נבחר מספר אקראי M ונמשוך קו מ-M בציר y עד שנתקל בגרף, הערך של X בנקודת מפגש זו הוא הערך של המשתנה האקראי X. נובע מזה, שאם המספר האקראי M הוא בין 0 ו-1/6, הערך המתאים של X הוא 1. אם הערך של M הוא בין 1/6 לבין 2/6, הערך המתאים של X הוא 2, וכו'. אפשר לתאר את היחס בין M ו-X באמצעות הביטוי הבא:

```
X:= TRUNC (6*M) +1
```

הפונקציה המתאימה לבעית הקוביה היא:

```
FUNCTION KUBIYA:INTEGER;
  BEGIN
     KUBIYA:= TRUNC (6 * RANDOM) +1
  END;
```

הערכים של המשתנה האקראי X (המספר של הפאה העליונה של הקוביה) הם בין 1 לבין 6. פונקצית ההסתברות שווה ל:

$f(x)=1/6 \qquad x=1,..,6$

אם נרשום a במקום 1 ונרשום b במקום 6, נוכל לכתוב את הנוסחא בצורה יותר כללית:

$$f(x) = \frac{1}{b-a+1} \qquad x = a, a+1, ..., b$$

עבור ערך טיפוסי של המשתנה X נקבל את הנוסחה הבאה:

```
X:= TRUNC ((b-a+1) * M) + a
```

במקום הנוסחא הקודמת:

```
X:= TRUNC (6*M) + 1
```

נוסחא זו תקפה עבור כל משתנה שיש לו הסתברות זהה בכל אחת מן הנקודות בתחום מ-a עד b. בעית זריקת הקוביה, כאשר a היה 1 ו-b היה 6, היא מקרה מיוחד של הנוסחא הכללית.

9.2.1 הרחבות

מקרה א:

כאשר משתנה אקראי X נע בתחום שבין a לבין b, דומה הדבר כאילו הוא נע בין a-1/2 לבין b+1/2 (עיין בגרף של פונקצית ההסתברות שהוצג קודם). לכן, בזמן קביעת פונקצית ההסתברות f(x) מחשבים את ההפרש הבא:

.(b+1/2)-(a-1/2)=b-a+1

כל זה נכון אם הערכים שהמשתנה האקראי מקבל הם שלמים כמו: a, a+1, a+2..b. אם המשתנה האקראי X מוגדר על פני טווח רציף כמו למשל, a≤X≤b, אז הגבולות a ו-b הם מדוייקים. הנוסחה הכללית ליצירת מספרים אקראיים מהתפלגות זו הופכת להיות: X:=(B-A)*M+A. כאשר M קרוב ל-0, X קרוב ל-a וכאשר M קרוב ל-1, נקבל X קרוב ל-b.

מקרה ב:

אפשר לדמות זריקה של שתי קוביות בשתי צורות:

א. נעשה דימוי של זריקת קוביה אחת פעמיים, ונחבר את התוצאות של שני הדימויים.

ב. נתייחס לעובדה שפונקציות ההסתברות וההתפלגות עבור זריקת שתי קוביות נראות כך:

x	f(x)	F(x)
2	1/36	1/36=.028
3	2/36	3/36=.083
4	3/36	6/36=.167
5	4/36	10/36=.277
6	5/36	15/36=.417
7	6/36	21/36=.583
8	5/36	26/36=.722
9	4/36	30/36=.833
10	3/36	33/36=.916
11	2/36	35/36=.972
12	1/36	36/36=1.0

נוכל ליצור משתנה אקראי M אחד לזריקה של שתי הקוביות ולחשב את X בדרך זו:

```
M:= RANDOM;
IF  M <= 0.028 THEN
  X:=2
ELSE
  IF  M <= 0.083 THEN
    X:=3
  ELSE
    IF  M <= 0.167 THEN
      X:=4
    ELSE
      IF  M <=0.277 THEN
        X:=5
      ELSE
```

```
IF  M <=0.417 THEN
  X:=6
ELSE
  IF  M <= 0.583 THEN
    X:=7
  ELSE
    IF  M <= 0.722 THEN
      X:=8
    ELSE
      IF  M <= 0.833 THEN
        X:=9
      ELSE
        IF  M <= 0.916 THEN
          X:=10
        ELSE
          IF  M <= 0.972 THEN
            X:=11
          ELSE
            X:=12;
```

מקרה ג:

אם רוצים לדמות משחק קלפים, צריכים לקחת בחשבון שבחירת קלף שונה מזריקת קוביה. בזריקת קוביה ההסתברות לקבל תוצאה כלשהיא היא תמיד 1/6, ללא תלות במספר הפעמים שזורקים את הקוביה קודם. אבל בבחירת קלפים, ההסתברות לקבל קלף מסוים תלויה במספר הקלפים שנמצאים בכל עת בחפיסה. למשל, ההסתברות לבחירת קלף מסוים היא 1/52 כאשר החפיסה שלמה ובה 52 קלפים, אבל אם ישנם רק 10 קלפים והוא אחד מהם, ההסתברות עולה ל-1/10.

השגרה הבאה מדמה טריפה של כל הקלפים בחפיסה. בשגרה זו בוחרים קלף אחד באופן אקראי ושמים אותו במקום האחרון. אז בוחרים קלף שני מתוך 51 שנשארו ושמים אותו במקום השני מן האחרון וכו'. אם רוצים למשל 5 קלפים אקראיים, אפשר להשתמש בשגרה זו ולבצע את הלולאה 5 פעמים במקום 51 פעמים. 5 הקלפים האחרונים בחפיסה יחשבו במקרה זה כ-5 קלפים שנבחרו באופן אקראי.

נגדיר את המערך CHAFISA שיחזיק את מספר הקלף שנמצא ברגע מסוים במקום I בחפיסה:

```
VAR
    CHAFISA: ARRAY[1..52] OF INTEGER;
```

במצב ההתחלתי, לפני טריפת הקלפים, אפשר לשים קלף I במקום I בחפיסה:

```
FOR I:= 1 TO 52 DO
   CHAFISA[I]:=I;
```

הפרוצדורה השלימה היא:

```
PROCEDURE TRIFA;
VAR
      I,J,ZMANI: INTEGER;
BEGIN
```

```
FOR  J:= 52 DOWNTO 2 DO
   BEGIN
     I:= TRUNC(J*RANDOM)+1;  (* בוחרים קלף אקראי בין 1 לבין J *)
                             (* שומרים את המספר הסידורי של הקלף הזה *)

     ZMANI:= CHAFISA[I];     (* מכניסים למקום I את הקלף שנמצא עתה במקום J בחפיסה *)
     CHAFISA[I]:= CHAFISA[J];
                             מכניסים את הקלף הנבחר לסוף החפיסה, *)
                             (* יחד עם הקלפים האחרים שנבחרו באופן מקרי.

     CHAFISA[J]:= ZMANI
   END
END.
```

9.3 פרמטרים שהם פונקציות או פרוצדורות

הפרמטרים של פונקציה או פרוצדורה יכולים להיות משתנים מכל הסוגים, כולל מערך והם יכולים גם להיות פונקציות או פרוצדורות.

כותבים פונקציה או פרוצדורה כפרמטר של פונקציה, בעיקר כאשר כותבים פרוצדורה לחישוב כללי במתמטיקה, כמו למשל אינטגרציה, או ציור של גרף. כדי שפרוצדורה מסוג זה תהיה כללית, עליה לפעול על כל פונקציה ולא רק על פונקציה מסוימת. משיגים זאת כאשר כותבים פרמטר כללי שבזמן הביצוע מחליפים אותו בפונקציה שתבצע את הפעולה המתמטית המבוקשת.

<u>ציור גרף של שתי פונקציות:</u>

```
PROGRAM TZIYUR (OUTPUT);
FUNCTION SINUS (X: REAL): REAL;
  BEGIN
    SINUS:= SIN(X)
  END;
FUNCTION  COSINUS (X: REAL): REAL;
    BEGIN
      COSINUS:= COS(X)
    END;

PROCEDURE  GRAPH (FUNCTION F, G: REAL; MAX, MIN: REAL);
  CONST
     GODEL:=100;
  VAR
     TVACH, Y: REAL;
     MINIMUM, MAXIMUM, X, A, B, C, I: INTEGER;

   PROCEDURE  MINMAX (A, B: INTEGER;
                     VAR  MINIMUM, MAXIMUM: INTEGER);
    BEGIN
      IF  A<B  THEN
```

```
(20)            BEGIN
(21)              MINIMUM:=A;
(22)              MAXIMUM:=B
(23)            END
(24)          ELSE
(25)            BEGIN
(26)              MINIMUM:=B;
(27)              MAXIMUM:=A
(28)            END
(29)      END;

(30)    BEGIN                              (* GRAPH הפרוצדורה *)
(31)       TVACH:= MAX - MIN;
(32)       FOR  X:=1  TO 50  DO
(33)         BEGIN
(34)           Y:=0.1*X;
(35)           A:= ROUND (((F(Y) - MIN) / TVACH) * GODEL);
(36)           B:= ROUND (((G(Y) - MIN) / TVACH) * GODEL);
(37)           MINMAX (A, B, MINIMUM, MAXIMUM);
(38)           FOR  I:= 0 TO MINIMUM-1 DO
(39)             WRITE ('^');
(40)           WRITE ('*');
(41)           FOR  I:= (MINIMUM+1)  TO  (MAXIMUM-1)  DO
(42)             WRITE ('^');
(43)           IF  MINIMUM <> MAXIMUM  THEN
(44)             WRITE ('*');
(45)           WRITELN
(46)         END
(47)    END;

(48)    BEGIN                              (* תכנית ראשית *)
(49)       GRAPH  (SINUS, COSINUS, 1, -1)
(50)    END.
```

<u>הסבר:</u>

(48)-(50) התכנית הראשית: פקודת הפרוצדורה <u>GRAPH</u> נמצאת בשורה (49) ובה רושמים כפרמטרים שתי פונקציות: SINUS ו-COSINUS. לא השתמשנו בפונקציות SIN, COS אשר מוגדרות בשפת פסקל, כי יש מהדרים אשר לא מתירים שפונקציות מוגדרות תהינה פרמטרים. לכן, בשורות (2) עד (9) הגדרנו מחדש פונקציות אלו.

(10) הפרוצדורה GRAPH בשורה (10) מקבלת שתי פונקציות בעלות שמות דמה: F ו-G. בהצהרת הפרוצדורה אין צורך להתייחס לפרמטרים של פונקציות אלה, אלא רק לסוג הנתונים שהם מייצגים. כאשר משתמשים בפונקציות בתוך הפרוצדורה יש לכלול את הארגומנטים המתאימים.

כאשר מדפיסים גרף צריך לדאוג לכך שהסקלה שלו תתאים לסקלה של הפונקציה שמציירים. כלומר, שהערכים לא ישתרעו למשל בין 1 לבין 10,000, או בין 0.1 לבין 0.101. כדי להבטיח שהסקלה תהיה בטווח הרצוי, מחשבים את הטווח

(MAX-MIN) של הפונקציה (כאן נקבל 2=(1-)-1) ומחלקים בו את F(Y)-MIN.
התוצאה חייבת להיות פחות מ-1. אם נכפול אותה בגודל הרצוי של הגרף
(GODEL) נקבל מספר בין 0 ל-GODEL.

(34) משרטטים פונקציות F(y) ו-G(y) כאשר y נע בין 0.1 ל-5.0.

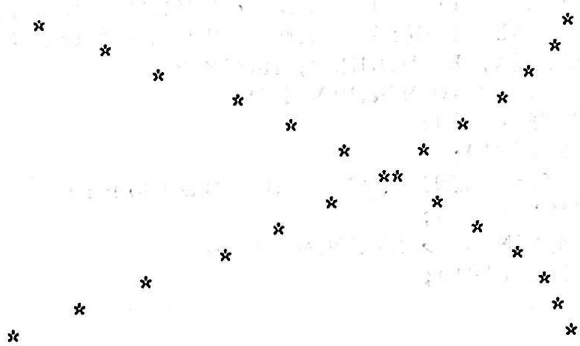
(37) בודקים איזה מבין המספרים יותר קטן.

(38)-(39) מדפיסים רווחים עד הערך של הפונקציה הקטנה.

(40) מדפיסים כוכבית כדי לסמן את ערכה של הפונקציה היותר קטנה.

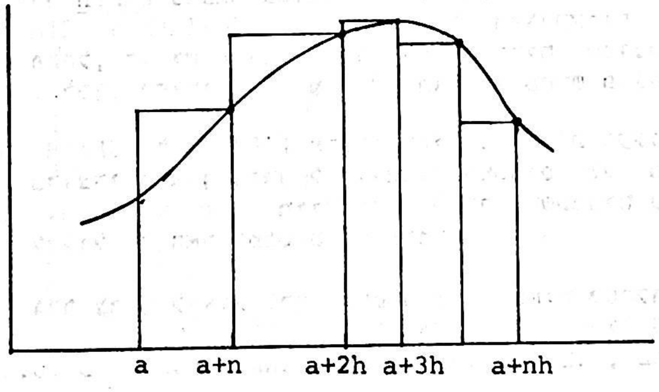
(41)-(42) מדפיסים שוב פעם כוכבית, כדי לסמן את הפונקציה היותר גדולה,
אם שתי הפונקציות אינן שוות.

פלט של תכנית זו נראה כך:



__פונקציה לביצוע אינטגרציה:__

פרוצדורה אחרת שפרמטרים פונקציונליים מתאימים לה היא פונקציה לביצוע
אינטגרציה. נניח שיש פונקציה f(x). נוכל לאמוד את שטחה בין הנקודות A
ו-B, אם נחלק את השטח ל-N מלבנים, כאשר הפינה הימנית של כל מלבן נמצאת על
העקומה. עיין בציור למטה:



אם $h=\frac{b-a}{n}$ ניתן לאמוד את השטח מתחת לעקומה ע"י המשוואה הבאה:

$$I = \sum_{i=1}^{n} f(a+ih) * h = h \sum_{i=1}^{n} f(a+ih)$$

נכתוב פונקציה בשפת פסקל שמבצעת אינטגרציה לפי שיטה זו.

```
FUNCTION INTEGRAL (FUNCTION F: REAL; A, B: REAL;
                                    N: INTEGER) : REAL;
VAR
   SCHUM, H: REAL;
   I: INTEGER;
BEGIN
  H:= (B-A)/N;
  SCHUM:=0;
  FOR  I:=1  TO  N  DO
     SCHUM:= SCHUM + F(A+I*H);
  INTEGRAL:= SCHUM*H
END;
```

כדי לאמוד את השטח של רבע העיגול בין 0 ל-R עבור y>0 נשתמש בפונקציה הבאה:

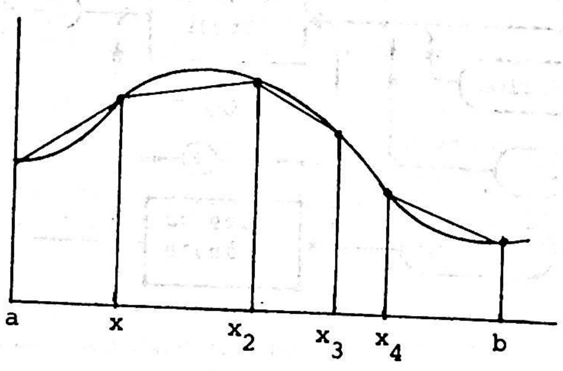
```
FUNCTION EGUL (X:REAL):REAL;
  BEGIN
    EGUL:= SQRT (SQR(R)-SQR(X))
  END;
```

הפקודה הבאה תדפיס את האומדן עבור השטח של העיגול כולו:

```
WRITE (4 * INTEGRAL (EGUL, 0, R, N))
```

ככל ש-N יותר גדול, האומדן יהיה יותר מדויק.

נוכל לשפר את האומדן ע"י שימוש בשיטת הטרפז, כלומר ע"י חלוקת השטח לטרפזים לשם חישוב השטח. שיטה זו תיתן אומדן יותר מדויק, שהרי טרפז תואם את העקומה ביתר דיוק ממלבן, כמו שרואים מן הציור הבא:



השטח של כל טרפז הוא: $\frac{f(x_i) + f(x_{i+1})}{2} * h$

כאשר $h=\frac{b-a}{n}$ הוא הרוחב של כל טרפז וגם $X_0=a$, $X_n=b$ נוכל לכתוב:

$$\int_a^b f(x)\,dx = \sum_{i=0}^{n-1} (h/2)\,[f(x_i) + f(x_{i+1})]$$

$$= (h/2)\,[(f(x_0)+f(x_1)) + (f(x_1)+(f(x_2)) + \ldots + (f(x_{n-1})+f(x_n)]$$

$$= (h/2)\,[f(x_0) + 2f(x_1) + \ldots + 2f(x_{n-1}) + f(x_n)]$$

$$= (h/2)\,(f(a)+f(b)) + h\sum_{i=1}^{n-1} f(a+ih)$$

כדי להפוך את שיטת החישוב של הפונקציה INTEGRAL משיטה מלבנית לשיטה טרפזית, נחוצים רק השינויים האלה:

א. בשורה (8) נחליף את SCHUM:=0 בפקודה הבאה:

```
SCHUM:= 0.5*H * (F(A) + F(B))
```

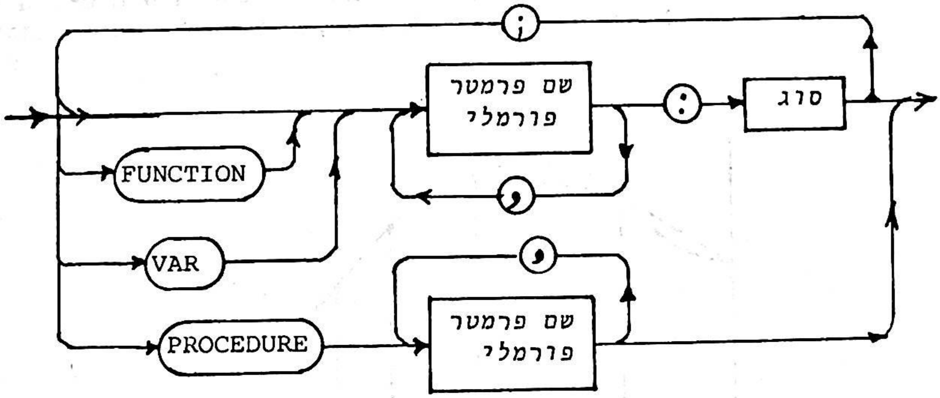
ב. בשורה (9), במקום FOR I:=1 TO N נכתוב:

```
FOR I:= 1 TO N-1
```

כאשר פונקציות או פרוצדורות הן ארגומנטים, הארגומנטים של אותן פונקציות או פרוצדורות חייבים להיות פרמטרים ערכיים סקלריים.

<u>שמות הפרמטרים הפורמליים וסוגיהם:</u>

לסיכום, נוכל לתת גירסה מורחבת של תיאור "שמות הפרמטרים הפורמליים וסוגיהם", שבהגדרת פונקציה ופרוצדורה ונשתמש למטרה זו בתרשים הבא:



הסבר:

בתרשים זה נוספו המלים FUNCTION ו-PROCEDURE. אחרי המלה VAR יבוא שם של פרמטר פורמלי, שהוא שם של פרמטר משתנה. אחרי המלה FUNCTION יבוא שם של פרמטר פורמלי שהוא שם פונקציה. אפשר לכתוב שם של פרמטר פורמלי בלי אף אחת מן המלים VAR ו-FUNCTION ואז הכוונה לפרמטר ערכי. בכל אחד משלושת המקרים האלה, אחרי השם של הפרמטר הפורמלי צריכים לכתוב את הסוג שלו.

האופציה הרביעית היא לכתוב את המלה PROCEDURE ואחריה שם של פרמטר פורמלי, שהוא שם פרוצדורה. במקרה הזה, לא צריכים לציין סוג, כי השם של הפרוצדורה לא מיוחס לסוג כלשהוא, כי רק הפרמטרים שלה יכולים להיות מיוחסים לסוג מסוים. פרמטרים אלה לא מופיעים כחלק מן ההצהרה כאשר הפרוצדורה היא בעצמה רק פרמטר, אלא במקום אחר בהצהרת אותה פרוצדורה.

9.4 פרוצדורות לשליטה במסוף

בסעיף זה נדון בפרוצדורות, אשר תומכות בפעולות קלט ופלט במסוף ושאינן פעולות קריאה וכתיבה. הן מנצלות את התכונות המיוחדות של המסופים לנוחות המשתמש במסוף.

כאשר כותבים פקודת WRITE בתכנית, המחשב מעביר למסוף רצף של סיביות אשר מייצגות את התווים אשר מיועדים להצגה. כל קבוצה של 8 סיביות מתורגמת לתווים לפי קוד ASCII או EBCDIC, בהתאם לתכונות המסוף. רוב התווים מוצגים במסוף כמו למשל, האות D או הסיפרה 4. יש מיעוט של תווים שלא מוצגים כלל על המסוף, מכיון שהם מיועדים לבקרה בלבד. לשם כך על המסוף להיות בעל כושר לפענח קודים מסוג זה ולבצע את הפעולות הנובעות מהם.

כדי שאפשר יהיה לחבר מסופים של יצרנים שונים למחשבים שונים היה צורך בתקן אחיד לתווים שמוצגים ולתווי הבקרה. התקנים הנפוצים הם תקן ASCII ותקן EBCDIC.

המסך של המסוף הנפוץ מכיל 24 שורות ו-80 עמודות ולכן הקיבולת שלו יכולה להגיע ל-1920 תווים. כדי למקם את <u>הסמן</u> (Cursor) במקום כלשהו על פני המסך, יש לציין את הקואורדינטות X,Y כלומר, את השורה ($1 \le Y \le 24$) ואת העמודה ($1 \le X \le 80$). למשל, אם רוצים למקם את הסמן בשורה 7 ובעמודה 27, ניתן ערך 7 ל-Y וערך 27 ל-X ונשלח נתונים אלו למסוף בפקודת WRITE.

על המסוף לדעת להבחין בין תווי בקרה, במקרה זה - לשם הצבת הסמן, לבין תווים אחרים שמיועדים להצגה. ההבחנה נעשית ע"י כך ששולחים למסוף <u>תווי קוד</u> (Escape Sequence) לפני תווי הבקרה. זוהי סדרת תווים מיוחדים שמורים למחשב שהתווים שיבואו אחריהם מיועדים לתפקיד מסוים ולא להצגה רגילה. אחרי סדרת תווי הבקרה יירשם תו מיוחד כדי לציין את סוף הסדרה, כפי שנראה בהמשך. ההוראה ניתנת בפקודת WRITE.

נציג עתה כמה פרוצדורות אשר מבצעות פונקציות שימושיות במסופים ומשתמשות בתווי בקרה המומלצים ע"י <u>ANSI</u> (American National Standard Institute).

המתכנת חייב לבדוק את ההתאמה והיישימות של הפרוצדורות והקודים המוצגים כאן לסביבת העבודה שלו, אשר כוללת מחשב, מסופים ומהדר פסקל.
את פקודות WRITE השונות יש לכתוב כפי שמוצג בדוגמאות.

9.4.1 מחיקת מסך

פרוצדורה זו מאפשרת מחיקה מלאה או חלקית של המסך, על פי תווי הבקרה שהיא מקבלת.

```
(1)   PROCEDURE MASACH (Y, X: INTEGER);
(2)     BEGIN
(3)       IF  Y < 0  THEN
(4)         BEGIN
(5)           WRITE (CHR(155), '[2J');      (* מחק את כל המסך *)
(6)           Y:= -Y
(7)         END;
(8)       WRITE (CHR(155), '[', Y:1, ';', X:1, 'H')     (* הצב סמן *)
(9)     END;
```

<u>הסבר:</u>

(1) הפרוצדורה מקבלת את הקורדינאטות (Y,X) של מיקום הסמן. תחילה כותבים את Y (השורה) ואח"כ - את X (העמודה).

(3) מפענחים את הסימן השלילי של Y כבקשה למחוק את המסך בהתאם לקוד הפעולה.

(5) הפרוצדורה שולחת למסך את תווי הקוד ע"י ההוראה:('[',(WRITE(CHR(155.
לאחריה כותבים 2J שהוא הקוד למחיקה של כל המסך. לפעולות מחיקה שונות נדרשים קודים שונים, כפי שמפורט ברשימה שלהלן:

0J - מחיקה מהמקום שבו נמצא הסמן עד לסוף המסך.
1J - מחיקה ממשבצות (1,1) עד המקום הנוכחי של הסמן.
2J - מחיקת כל המסך.
0K - מחיקה מן הסמן עד לסוף השורה.
1K - מחיקה מהתחלת השורה עד לסמן.
2K - מחיקת כל השורה שבה נמצא הסמן.

אם רוצים למשל, למחוק את המסך רק מן הנקודה שבה נמצא הסמן, אפשר לכתוב בשורה (5) של התכנית: WRITE (CHR(155), '[0J');

(6) מקנים ל-Y ערך חדש, חיובי, לשם מיקום הסמן.

(8) התווים (CHR(155 ו-'[' מהווים את הקוד שמורה למחשב שהערכים של X ושל Y שאחריהם מיועדים למיקום הסמן. התו 'H' מציין את סוף הסדרה של תווי הבקרה.

9.4.2 הבהוב תווים

במסופים רבים קיימת התכונה של <u>הבהוב</u> (Blink). פרוצדורה זו גורמת להפעלת הבהוב תווים שיוצגו על המסך. להפסקת ההבהוב יש לתת הוראה אחרת.

```
(1)   PROCEDURE HIVHUV (KETZEV: INTEGER);
(2)     BEGIN
(3)       WRITE (CHR(155), '[', KETZEV: 1, 'v');
(4)       WRITE (CHR(155), '5m')
(5)     END;
```

<u>הסבר:</u>

הפרמטר KETZEV קובע את קצב ההבהוב. במסוף מסוים, הוא נע בתחום מ-0 (המהיר ביותר) כאשר אורכו של כל הבהוב הוא 0/30 שניות ועד ל-255 (האיטי ביותר) כאשר אורכו של כל הבהוב הוא 255/30 שניות.

לאחר שקובעים את הקצב, כלומר נותנים ערך למשתנה KETZEV מפעילים את ההבהוב בשורה (4). אם לא קובעים את הקצב, הפרוצדורה משתמשת בברירת המחדל שהיא לדוגמא, 32/30 שניות (ז"א KETZEV=32).

9.4.3 הבלטה של תצוגת התווים

כאשר עובדים במסוף יש צורך לעתים להדגיש תצוגה של תווים כדי למשוך את תשומת הלב של מפעיל המסוף כמו למשל, לשם הוראת פעולה או הודעת שגיאה, הצגת כותרת של טופס ועוד. בפרוצדורה זו מאפשרים למשתמש להבליט את הפלט ולהציגו בצורות שונות. בנוסף להבהוב, אפשר לבצע ארבע פעולות נוספות.

```
PROCEDURE TECHUNOT (TEUR: CHAR);
  BEGIN
    CASE TEUR OF
      'R': WRITE (CHR(155), '[7m');     (* Reverse Video *)
      'U': WRITE (CHR(155), '[4m');         (* Underline *)
      'H': WRITE (CHR(155), '[2m');    (* Half Intensity *)
      'G': WRITE (CHR(155), '[10m');         (* Graphics *)
      OTHERWISE WRITELN ('טעית בפרמטר')
    END
  END;
```

<u>הסבר:</u>

הפעולה המבוקשת ע"י המתכנת נמסרת לפרוצדורה ע"י הפרמטר המשתנה TEUR. האפשרויות הן:

R - Reverse Video - התווים יוצגו על המרקע בהיפוך של צבע. למשל, אם התצוגה הרגילה היא ירוק על מקרע שחור, נקבל עתה תצוגה של תווים בשחור על רקע ירוק של השורה.

U - Underline - מתחת לכל תו שיופיע במסך ישורטט קו.

H - Half Intensity - התווים שיופיעו מכאן ואילך יהיו בחצי הבהירות הרגילה.

G - Graphics - במקום אותיות לטיניות קטנות, יופיעו סמלים גרפיים שונים שמהם ניתן לבנות במסך ציורים וצורות שונות.

9.4.4 חזרה לתצוגה הבסיסית

בפרוצדורה זו "מכבים" את התכונות שהופעלו וחוזרים למצב הסטנדרטי.

```
(1)   PROCEDURE KEBUY (TEUR: CHAR);
(2)   BEGIN
(3)    CASE TEUR OF
(4)      'T': WRITE (CHR(155), '[0m');     (* הפסק את כל התכונות *)
(5)      'R': WRITE (CHR(155), '[m');   (* Reverse Video סיים רק *)
(6)      'G': WRITE (CHR(155), '[11m');    (* סיים אופצית גרפיקה *)
(7)      OTHERWISE WRITELN ('טעות בפרמטר')
(8)    END
(9)   END;
```

הסבר:

(4) פרמטר T (עבור תכונות) מכבה את כל התכונות שהוזכרו בפרוצדורה הקודמת חוץ מן האופציה לגרפיקה. התכונות שהופעלו ע"י R, H או U יופסקו אם הפרמטר בפרוצדורה זו הוא T.

(5) אם רוצים לסיים רק את התכונה של Reverse Video ולהשאיר את התכונות האחרות פועלות, ניתן לעשות זאת בעזרת תו הבקרה M.

(6) כדי להחליף סמלים גרפיים באותיות לטיניות קטנות, חייבים לכתוב את הקוד 11m. הפעלת שורה (4) לא מפסיקה את האופציה של סמלים גרפיים.

9.5 תרגילים

9.5.1 שאלות

(1) עליך ליצור 100 קבוצות של 12 מספרים אקראיים, לחבר אותם בכל קבוצה ולצייר בגרף את ערכי 100 הסכומים. התוצאה צריכה להיראות כפעמון (עקומה נורמלית) ששיאו נמצא קרוב לערך 6 בציר ה-X.
אם במהדר פסקל שברשותך אין פונקציה סטנדרטית ליצירת מספרים אקראיים, תוכל להשתמש באחת מן הפרוצדורות שניתנו בפרק זה.

(2) בדוק אם הפרוצדורות לשליטה במסוף, אשר ניתנו בסעיף 9.4, מתאימות לסביבת העבודה שלך. אם לא, מהן הפרוצדורות המתאימות לעבודה במחשב שברשותך. פרט את הקודים לביצוע הפעולות השונות.

9.5.2 תכניות

(1) ערוך דימוי של זריקת שתי קוביות 1000 פעמים ושרטט היסטוגרמה בכדי להראות את השכיחות של כל תוצאה. השווה את השכיחויות היחסיות להסתברויות התאורטיות המוזכרות בפרק זה.

(2) ערוך הדמייה של המשחק Craps, אשר משחקים אותו לפי נוהל זה: אחד מטיל שתי קוביות. אם הסכום שמתקבל הוא 7 או 11 הוא זוכה. אם הסכום הוא 2, 3, או 12 הוא מפסיד ומעביר את הקוביות לחברו. אם הסכום הוא כל מספר אחר, המשחק ממשיך להטיל את הקוביות עד שאחד משני דברים קורה:

- הוא מקבל שוב פעם את אותו סכום שקיבל בתחילת המשחק וזוכה.
- הוא מקבל 7 ואז מפסיד ומעביר את הקוביות.

א. כתוב תכנית שתאפשר לאדם להמר על כל משחק. אם הוא זוכה המחשב חייב לו, אחרת הוא חייב למחשב. התכנית תחשב את מאזן הזכיות או ההפסדים לאחר סדרה של משחקים.
ב. ערוך הדמייה של 1000 משחקים ובדוק מהי ההסתברות שהשחקן יזכה (הערך התיאורטי הוא 0.4929).

(3) עשה אינטגרציה של הפונקציה $Y=X^2$ בין 0 ל-1, ע"י בחירה של מספרים אקראיים בתחום שבין 0 ל-1. קבע את האחוז של הנקודות שתהיינה בין ציר X לבין העקומה, כאשר כל זוג של מספרים מייצג זוג (X,Y).

(4) את המשחק "21", שנקרא "קוויטלד" (באידיש) או "פתקים", נהוג היה לשחק בחג החנוכה.
נוהל המשחק: מחלקים קלף לכל אדם. כאשר מגיע תורו, הוא בוחר קלף מן החפיסה. אם אחד המתחרים מגיע לסכום של 21 הוא זוכה מיד, אם הסכום גבוה מ-21 הוא מפסיד מיד. אם הסכום שלו קטן מ-21, יש לו גם זכות לא לבחור קלף. אם בסיום הסיבוב אף לא אחד קיבל 21, מגלים את הקלפים ומי שבידו הסכום הגבוה ביותר אשר קרוב ל-21 הוא הזוכה. דמה את המשחק הזה בעבור אדם שמשחק נגד המחשב ותוכל להניח את הפרטים הנוספים:

א. אתה יכול להמר לפני כל משיכת קלף ולהוסיף להימור הקודם.
ב. המחשב לא מהמר. הוא רק משלם לך אם הימרת וזכית.
ג. המחשב תמיד מושך עוד קלף, אם סכום הקלפים שלו פחות מ-17.
ד. הקלפים 2-9 מייצגים את המספרים 2-9.
ה. הקלף "אס" מייצג את המספר 1 או 11 לפי רצון השחקן.
ו. הערך של כל אחד מהקלפים "ז'ק", מלכה ומלך הוא 10.

(5) כתוב פונקציה שמקבלת כקלט שתי פונקציות שונות, אשר משתמשות באותו ערך ממשי X וקובעת איזו מביניהן יותר גדולה בערכה המספרי. הפונקציה מחזירה את הערך של הפונקציה היותר גדולה ומציינת איזו מהן היא הגדולה.

(6) כתוב פונקציה בכדי לעשות אינטגרציה נומרית של כל פונקציה לפי שיטת הטרפז. השתמש בפונקציות הבאות כדי לבדוק אם הפונקציה שכתבת פועלת כראוי.

א. $\int_0^1 SIN(x)\, dx$

ב. $\int_{100}^{200} x^2\, dx$

ג. $\int_0^3 \sqrt{x}\, dx$

פרק 10
רשומות

10.1 מבנה רשומות

עד עכשיו עסקנו בנתונים מן הסוגים הבאים:

- מספרים ממשיים,
- מספרים שלמים,
- ערכים בוליאניים,
- תווים יחידים,
- סוגים שהוגדרו ע"י מנייה (Enumerated Types),
- מלים שהוגדרו כמערך ארוז (Packed Array).

לעתים עוסקים בקבוצה של נתונים שיש קשר לוגי ביניהם. אם כל נתון הוא מאותו סוג נוכל להחזיק את הקבוצה במערך ואם יש קבוצות רבות כאלו, אפשר להחזיק אותן במערך של מערכים. למשל, בתכנית MASKORET13 החזקנו כך רשימה של שמות עובדים:

```
TYPE SHEMOVED = PACKED ARRAY [1..ORECH] OF CHAR;
VAR OVDIM : ARRAY [1..MAXOVDIM] OF SHEMOVED;
```

נוכל להחזיק ציונים של 20 תלמידים בכל אחד מ-5 מבחנים בצורה זו:

```
TYPE TZIYUNIM = ARRAY [1..5] OF INTEGER;
VAR STUDENTIM : ARRAY [1..20] OF TZIYUNIM;
```

כל קבוצת נתונים במקרה של העובדים, היתה מורכבת ממספר תווים לפי ערכו של הקבוע ORECH. כל קבוצת נתונים במקרה של הסטודנטים היתה מורכבת מ-5 מספרים שלמים.בשני המקרים מרכיבי הקבוצה זהים בסוגם.

לעתים יש צורך לטפל בקבוצה של משתנים שמרכיביה הם מסוגים שונים. במקרה של העובדים, אין זה נכון להניח שהרשימה כוללת שמות בלבד. יותר סביר להניח שיש ברשימה נתונים שונים עבור כל עובד: מספר עובד, שם, תאריך לידה, כתובת, מצב משפחתי, שכר או מרכיבי שכר וכד', אשר כל אחד מהם מוצג כסוג שונה. קבוצת נתונים כזו נקראת רשומה (Record) וכל איבר בה נקרא שדה (Field). השדות ברשומה אינם חייבים להיות מאותו סוג נתונים. קבוצת רשומות העוסקות בנושא מסוים נקראת קובץ (File), כמו למשל קובץ תלמידים.

דוגמא של רשומה מוצגת בשרטוט הבא:

שם עובד	שכר	ציון

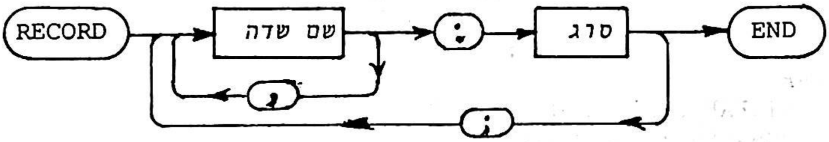
בתכנית MASKORET13 שבפרק 7, טיפלנו בשדות מסוגים שונים ע"י כך שבנינו 3 מערכים שונים: SCHAR ,OVDIM, TZIYUN. המקום היחסי המקביל בכל אחד מ-3 המערכים התייחס לאותו עובד. נשכתב עתה תכנית זו למקרה שכל הנתונים של עובד מסוים נכללים ברשומה אחת ונשתמש במערך אחד של רשומות.

```
PROGRAM MASKORET16 (INPUT, OUTPUT);
CONST
  MAXOVDIM=100;
  ORECH=25;
TYPE
  RESHUMAT_OVED= RECORD
                  SHEMOVED: PACKED ARRAY [1..ORECH] OF CHAR;
                  SCHAR: REAL;
                  TZIYUN: CHAR
                 END;
VAR
  MISPAR_OVDIM : 0..MAXOVDIM;
  MCHAPES : BOOLEAN;
  MARACH_OVDIM : ARRAY [1..MAXOVDIM] OF RESHUMAT_OVED;
  RESHUMA_CHADASHA : RESHUMAT_OVED;
  MAKOM, MISPAR, MISPAR_CHADASH : 1..MAXOVDIM;
BEGIN
  MISPAR_OVDIM:= 0;
  WHILE NOT EOF AND (MISPAR_OVDIM < MAXOVDIM) DO
   BEGIN
    WRITELN ('הקש שם עובד, שכר וציון ');
    FOR MAKOM:= 1 TO ORECH DO
      READ (RESHUMA_CHADASHA.SHEMOVED[MAKOM]);
    READLN (RESHUMA_CHADASHA.SCHAR, RESHUMA_CHADASHA.TZIYUN);
    MISPAR:= 1;
    MCHAPES:= TRUE;
    WHILE MCHAPES AND (MISPAR <= MISPAR_OVDIM) DO
      IF RESHUMA_CHADASHA.SHEMOVED <
                           MARACH_OVDIM[MISPAR].SHEMOVED THEN
         MCHAPES:= FALSE
      ELSE
         MISPAR:= SUCC(MISPAR);
    MISPAR_CHADASH := MISPAR;
    FOR MISPAR := MISPAR_OVDIM DOWNTO MISPAR_CHADASH  DO
       MARACH_OVDIM[SUCC(MISPAR)] := MARACH_OVDIM[MISPAR];
    MARACH_OVDIM[MISPAR_CHADASH] := RESHUMA_CHADASHA;
```

```
(37)      MISPAR_OVDIM := SUCC(MISPAR_OVDIM);
(38)     END;
(39)     FOR MISPAR:=1 TO MISPAR_OVDIM DO
(40)       BEGIN
(41)         WRITE (MARACH_OVDIM[MISPAR].SHEMOVED,
(42)                 MARACH_OVDIM[MISPAR].SCHAR:10:2);
(43)         CASE MARACH_OVDIM[MISPAR].TZIYUN  OF
(44)           'A': WRITELN ('METZUYAN':10);
(45)           'B': WRITELN ('TOV':05);
(46)           'C': WRITELN ('MEMUTZA':08)
(47)         END
(48)       END
(49)  END.
```

<u>הסבר:</u>

(5)-(10) בשורות אלו מצהירים על הרשומה RESHUMAT_OVED כסוג המוגדר ע"י המשתמש. ההצהרה חייבת להתחיל במלה RECORD ולהסתיים במלה END, כאשר בין מלים אלו מופיעים שמות השדות המרכיבים את הרשומה. הצורה הכללית של ההצהרה היא:



אפשר לכתוב שדות אחדים מאותו סוג ולהפריד ביניהם בפסיק. בסוף כל קבוצה של שדות מאותו סוג יבואו נקודתיים, שם הסוג ונקודה-פסיק. אח"כ אפשר להוסיף שדות מסוגים אחרים. בדוגמא זו הגדרנו את הרשומה בהצהרת TYPE, אך אפשר גם להגדיר אותה כמשתנה בהצהרת VAR.

(14)-(15) אחרי ההצהרה של RESHUMAT_OVED אפשר להשתמש בה בהצהרת משתנים: בתיאור איברי מערך (שורה 14) ובתיאור של משתנה בודד (שורה 15) שיקרא RESHUMA_CHADASHA.

(23)-(24), (43)-(41) מתייחסים לשדה בתוך הרשומה ע"י ציון שם הרשומה, נקודה ואחריה שם השדה כמו למשל, RESHUMA_CHADASHA.SCHAR (שורה (24)).
אם השדה הוא מערך שמעוניינים באחד מאבריו, כותבים את האינדקס בתוך סוגריים מרובעים אחרי שם השדה, למשל (שורה 23):

RESHUMA_CHADASHA.SHEMOVED[MAKOM]

אם הרשומה עצמה שייכת למערך, כותבים את שם המערך, מספר הרשומה במערך בתוך סוגריים מרובעים, נקודה ואח"כ שם השדה כמו למשל: MARACH_OVDIM[MISPAR].SHEMOVED (שורה 41). עיין גם בשורות (42)-(43).

בשורות (22)-(24) חוזרים על שם הרשומה (RESHUMA_CHADASHA) 3 פעמים. אפשר לפשט את הקידוד ע"י החלפת שורות אלו בפקודות הבאות:

```
WITH RESHUMA_CHADASHA DO
  BEGIN
```

```
    FOR MAKOM:= 1 TO ORECH DO
      READ (SHEM_OVED[MAKOM]);
    READLN (SCHAR, TZIYUN)
  END;
```

כאשר כותבים שם רשומה בין WITH ו-DO, כל שם שדה שמופיע בפקודה הפשוטה או המורכבת שאחרי WITH מיוחס לאותה רשומה. לכן, כתיבת שם הרשומה, RESHUMA_CHADASHA, בין WITH לבין DO איפשרה לנו להשמיט את שם הרשומה שלפני SHEM_OVED, SCHAR ו-TZIYUN בפקודות העוקבות.

(35)-(36) המניע העיקרי להגדרת המושג של רשומה הוא העובדה שאפשר להעתיק מרשומה אחת לחברתה בעזרת פקודת הצבה, אם שתי הרשומות זהות במבנה שלהן. בתכנית MASKORET13 היינו מוכרחים לבצע העברה זו בנפרד לכל שדה ולכן פעולה זו נכתבה שם ב-3 שורות ((36)-(38)). שורה (36) בתכנית זו גם משלבת 3 שורות שהיו נפרדות בתכנית MASKORET13 (שם, שורות (42)-(40)).

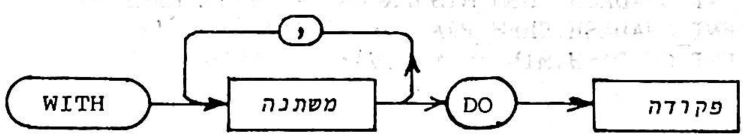
(39)-(48) גם שורות אלו ניתן לפשט כדלהלן:

```
FOR MISPAR:= 1 TO MISPAR_OVDIM DO

    WITH MARACH_OVDIM[MISPAR] DO
      BEGIN                                  (* WITH *)
        WRITE (SHEMOVED, SCHAR:10:2);
        CASE TZIYUN OF
          'A' : WRITELN ('METZUYAN':10);
          'B' : WRITELN ('TOV':5);
          'C' : WRITELN (MEMUZA':9)
        END                                  (* CASE *)
      END                                    (* WITH *)
```

10.2 הפקודה WITH..DO (*)

התבנית אשר מתארת את הפקודה WITH..DO מוצגת להלן:



המשתנה בהגדרות VAR חייב להיות שם של רשומה.

נשתמש בדוגמאות כדי להציג רשומות יותר מורכבות מאלו שראינו עד כה ונשתמש גם בפקודות WITH..DO.

ניהול ביקורים במרפאה:

במרפאה מחזיקים רשומה אחת עבור כל אחד מן החולים ("פציינטים"), אשר כוללת פרטים אישיים. יש בה התייחסות לרשומות אחרות שמחזיקות את המועד (תאריך ושעה) של הביקור ("תור") הבא. הרשומות והגדרות סוגי הנתונים ניתנים להלן:

```
TYPE
   MACHROZET_10 = PACKED ARRAY[1..10] OF CHAR;     (* 10 מקומות *)
   SUG_TARICH = RECORD
                  YOM     : 1..31;
                  CHODESH: ARRAY[1..6] OF CHAR;
                  SHANA   : 1985..1995
                END;
   TOR =        RECORD
                  TARICH: SUG_TARICH;
                  SHAA   : 0..2400
                END;
   PATIENT =    RECORD
                  SHEM_MISHPACHA      : MACHROZET_10; (* 10 מקומות *)
                  SHEM_PRATI          : MACHROZET_10;
                  TOR_HABA            : TOR;
                  TORIM_BEAVAR        : ARRAY[1..52] OF TOR;
                  MIN                 : (ZACHAR,NEKEVA);
                  GIL                 : 15..85;        (* רק מבוגרים *)
                  MATZAV_MISHPACHTI : (RAVAK,NASUI,ALMAN,GARUSH);
                  PATIENT_PRATI       : BOOLEAN;
                  ROFEH               : MACHROZET_10;
                  DIAGNOZOT           : ARRAY[1..10] OF MACHROZET_10
                END;
```

לאחר שמגדירים סוג עבור PATIENT אפשר להצהיר על משתנים מאותו סוג:

```
VAR
   PATIENT_CHADASH: PATIENT;
   PATIENTIM_SHEL_LEVI: ARRAY[1..10] OF PATIENT;
```

בתכנית יכולים לתת ערכים למשתנים. למשל, בזמן קבלת חולה חדש, נוכל לרשום פרטים בסיסיים, כמו,

```
PATIENT_CHADASH.SHEM_MISHPACHA := 'YITZCHAKI^';
PATIENT_CHADASH.SHEM_PRATI := 'SARA^^^^^^';
PATIENT_CHADASH.MIN := NEKEVA;
PATIENT_CHADASH.GIL := 26;
```

בזמן קביעת התור הראשון נוכל לכתוב:

```
PATIENT_CHADASH.TOR_HABA.TARICH.YOM := 17;
PATIENT_CHADASH.TOR_HABA.TARICH.CHODESH := 'AUGUST';
PATIENT_CHADASH.TOR_HABA.TARICH.SHANA := 1986;
PATIENT_CHADASH.TOR_HABA.SHAA := 1100;
```

במציאות מזינים נתונים חדשים מן המסוף ולא כותבים אותם כחלק אינטגרלי של

התכנית. במקום פקודות ההצבה המופיעות למעלה, צריך לכתוב פקודות READ או READLN, כולל שמות המשתנים והתיאור שלהם.
ההתייחסויות לשדות בפקודות ההצבה היו מסורבלות למדי ולכן נשתמש בפקודה WITH כדי לפשט את התכנות, כפי שנראה בהמשך ב-6 מבנים שונים:

<u>מבנה 1:</u>

```
(1)  WITH PATIENT_CHADASH DO
(2)      WITH TOR_HABA DO
(3)        WITH TARICH DO
(4)          BEGIN
(5)            YOM:= 17;
(6)            CHODESH:= 'AUGUST';
(7)            SHANA:= 1986;
(8)            SHEM_MISHPACHA:= 'YITZCHAKI^';
(9)            SHEM_PRATI:= 'SARA^^^^^^';
(10)           MIN:= NEKEVA;
(11)           GIL:= 26
(12)         END;
```

<u>הסבר:</u>
בדוגמא שלפנינו יש שלוש הוראות WITH מקוננות. סדר הסריקה שלהן הוא מן ההוראה הפנימית ביותר, בשורה (3), אל ההוראה החיצונית ביותר בשורה (1). המהדר בוחן תחילה את הפקודה WITH הפנימית בשורה (3) וקובע איזה מבין המשתנים המוזכרים שייכים לרשומה בשם TARICH. משתנים בשורות (5), (6) ו-(7) שייכים לרשומה זו ולכן הוא מצרף לפניהם את שם הרשומה.

המשתנים בשורות (8)-(11) אינם שייכים לרשומה TARICH ולכן הם לא מושפעים כלל מן העובדה שהם נמצאים בתוך פקודת WITH הפנימית.

עכשיו המהדר עובר לבחון את פקודת WITH האמצעית בשורה (2). TARICH.YOM, TARICH.CHODESH ו-TARICH.SHANA, שנבעו מהפעלת ה-WITH הפנימית, שייכים לרשומה TOR_HABA ולכן הוא מוסיף להם את שם הרשומה הנכון:

```
TOR_HABA.TARICH.YOM
TOR_HABA.TARICH.CHODESH
TOR_HABA.TARICH.SHANA
```

גם במקרה זה, כמו בקודם, המשתנים האחרים בטווח של ה-WITH האמצעית אינם שייכים לרשומה זו ולכן הם לא מושפעים.

לבסוף, המהדר מגיע לפקודה WITH החיצונית בשורה (1). כל המשתנים המוזכרים בשורות (5) עד (11) שייכים לרשומה בשם זה ולכן הן מתורגמות כפקודות הצבה לתוך הרשומה PATIENT_CHADASH.

יש להקפיד על סדר הפקודות של משפטי WITH. למשל נניח שמשנים את סדר הקינון של פקודות (1), (2), (3) וכותבים אותן בסדר שונה:

```
WITH  TOR_HABA DO
   WITH  TARICH DO
      WITH  PATIENT_CHADASH DO
```

במקרה זה, שורות (8)-(11) היו מתבצעות כראוי, אבל שורות (5)-(7) לא

היו פועלות כראוי, מכיון שהשדות YOM, CHODESH ו-SHANA אינם שייכים לרשומה PATIENT_CHADASH. אלה הם שדות של הרשומה TARICH, אשר מוגדרת כשדה של הרשומה TOR_HABA, שהיא עצמה שדה של הרשומה PATIENT_CHADASH. כאשר המהדר מגיע לרשומה TOR_HABA הוא דילג כבר על הרשומה PATIENT_CHADASH ואיננו יכול להתייחס אליה פעם נוספת.

נמצא, שפיתחנו את השדות בשורות (5)-(7) ל:

```
TOR_HABA.TARICH.YOM
TOR_HABA.TARICH.CHODESH
TOR_HABA.TARICH.SHANA
```

אבל לא הוספנו לפניהם את השם: PATIENT_CHADASH

ניתן לקצר את התכנית ובמקום לכתוב את השורות (1)-(3), אפשר לכתוב רק:

```
WITH  PATIENT_CHADASH, TOR_HABA, TARICH  DO
```

קיצור זה מסביר למה במבנה הכללי של WITH..DO היתה אפשרות לכתוב יותר ממשתנה (ז"א שם רשומה) אחד. המשמעות של המבנה המקוצר היא בדיוק כמשמעותו של המבנה הארוך יותר. לכן אסור לשנות גם פה את סדר המשתנים כמו שקודם היה אסור לשנות את סדר קינון הפקודות WITH..DO.

במקום מבנה 1 שהוזכר לעיל, אפשר לכתוב מבנים נוספים של סדרות פקודות אשר מכילות את הפקודה WITH..DO בצורות אחרות ואשר מבצעות את אותן הפעולות.

<u>מבנה 2:</u>

```
WITH  PATIENT_CHADASH DO
    BEGIN
       TOR_HABA.TARICH.YOM:= 17;
       TOR_HABA.TARICH.CHODESH:= 'AUGUST';
       TOR_HABA.TARICH.SHANA:= 1986;
       SHEM_MISHPACHA:= 'YITZCHAKI^';
       SHEM_PRATI:= 'SARA^^^^^^';
       MIN:= NEKEVA;
       GIL:= 26
    END;
```

<u>מבנה 3:</u>

```
WITH  PATIENT_CHADASH DO
   BEGIN
      MIN:= NEKEVA;
      GIL:= 26
      SHEM_MISHPACHA:= 'YITZCHAKI^';
      SHEM_PRATI:= 'SARA^^^^^^';
      WITH  TOR_HABA, TARICH DO
        BEGIN
           YOM:= 17;
           CHODESH:= 'AUGUST';
           SHANA:= 1986
        END
   END;
```

מבנה 4:

מבנה זה דומה למבנה 3, אבל במקום

```
WITH TOR_HABA,TARICH
```

כותבים:

```
WITH TOR_HABA.TARICH
```

ההבדל בין מבנה 3 לבין מבנה 4 הוא בכך שמבנה 3 מטפל בשדה שנמצא ברשומה TOR_HABA אפילו אם הוא לא נמצא ברשומה TARICH. מבנה 4 דורש שהשדות שיטופלו יהיו גם ב-TARICH וגם ב-TOR_HABA. בדוגמא זו המשתנים YOM, CHODESH ו-SHANA שייכים לשתי הרשומות ולכן גם מבנה 3 וגם מבנה 4 נכונים.

מבנה 5:

```
WITH  PATIENT_CHADASH DO
  BEGIN
    MIN:= NEKEVA;
    GIL:= 26;
    SHEM_MISHPACHA:= 'YITZCHAKI^';
    SHEM_PRATI:= 'SARA^^^^^^';
    WITH  TOR_HABA DO
      BEGIN
        TARICH.YOM:= 17;
        TARICH.CHODESH:= 'AUGUST';
        TARICH.SHANA:= 1986
      END
  END;
```

מבנה 6:

```
WITH  PATIENT_CHADASH DO
  BEGIN
    MIN:= NEKEVA;
    GIL:= 26;
    SHEM_MISHPACHA:= 'YITZCHAKI^';
    SHEM_PRATI:= 'SARA^^^^^^'
  END;
WITH  PATIENT_CHADASH.TOR_HABA DO
  BEGIN
    TARICH.YOM:= 17;
    TARICH.CHODESH:= 'AUGUST';
    TARICH.SHANA:= 1986
  END;
```

הנחייה:

מכל המבנים שהוצגו, הטובים ביותר הם מבנה 3 או מבנה 4. במבנה 1 ה-WITH..DO הכיל שלושה משתנים אשר שניים מתוכם (TOR_HABA ,TARICH) לא התאימו לחלק מן השדות בתחום ה-WITH..DO. לעומת זאת, במבנה 3 משתנה ה-WITH..DO החיצוני (PATIENT_CHADASH) שייך לכל השדות בתחומו ומשתני ה-WITH..DO הפנימי (TOR_HABA ,TARICH) שייכים גם כן לכל השדות בתחומם. מבנה 3 גם הרבה יותר ברור לקורא, שיכול לזהות ביתר קלות את הרשומות השונות והשדות השייכים לכל רשומה.

התחום של שם שדה הוא הרשומה שבה הוא מוגדר. נובע מזה שאותו שם שדה יכול להופיע ברשומות שונות מבלי ששם אחד ישפיע על חברו. רשומה יכולה להיות שדה בתוך רשומה אחרת ולכן גם שם רשומה יכול להופיע יותר מפעם אחת. התנאי לכך הוא שרשומות עם שמות זהים תהיינה שדות ברשומות שונות ובתנאי שהרשומה איננה שדה ברשומה אחרת בעלת שם זהה.

בפקודת WITH עבור רשומה מסוימת, אסור לפנות למשתנה שיש לו שם זהה לאחד מן השדות ברשומה. המהדר מצרף את שם הרשומה לכל איזכור של שדה שלה, אשר כלול בתחום ה-WITH..DO. אם יהיה משתנה אחר בעל אותו שם, גם אליו יצורף שם הרשומה, דבר שאיננו רצוי כמובן.

10.3 איתור מלות מפתח בתמליל (*)

יישום נפוץ במחשב הוא עיבוד תמליל (Text Processing). אחד מן העיבודים האפשריים הוא סריקת תמליל כדי לאתר את כל המופעים של מלה, או מלים, המוגדרות כמלות מפתח. סריקה כזאת מאפשרת להצביע על קטע תמליל אשר דן בנושא מסוים.

התכנית לעיבוד תמליל שבדוגמא זו מאתרת את כל השורות בתמליל (TEXT) שבהן מופיעה מלת מפתח (Key Word). היא מקבלת כקלט שלושה נתונים:

א. מספר מלות המפתח שרוצים לאתר,
ב. רשימת מלות המפתח,
ג. התמליל שרוצים לנתח.

התכנית מפיקה כפלט את התמליל המקורי ומציינת בשמאל של כל שורה את מספרה הסידורי. בסוף הקטע תרשמנה מלות המפתח ולצד כל אחת מהן ירשמו מספרי השורות שבהן היא הופיעה. נכתוב את התכנית בעבור תמליל באנגלית. היא תפעל גם בעברית לאחר שינויים קלים, אשר יקחו בחשבון גם את תכונות המסוף.

הרצה טיפוסית תיראה כך:

```
מספר מלות המפתח לתכנית זו יכול להיות
בין 1 ל-10^
3
עכשיו הקש את המלים:
COMPUTER
DATA
MACHINE
הכנס את הקטע

THE AUTOMATIC ELECTRONIC DIGITAL COMPUTER IS A MACHINE THAT USES
ELECTRONIC CIRCUITS TO MANIPULATE DATA EXPRESSED IN A SYMBOLIC FORM
ACCORDING TO SPECIFIC RULES IN A PREDETERMINED WAY. LET'S EXPLAIN
THIS IN DETAIL.
 FIRST OF ALL, THE COMPUTER IS A MACHINE.THIS MEANS IT CAN PERFORM
ONLY THOSE ACTIVITIES FOR WHICH THE BASIC CAPABILITIES HAVE BEEN
```

SPECIFICALLY DESIGNED INTO THE MACHINE.
SECONDLY, THE COMPUTER IS AUTOMATIC. THIS MEANS THAT ONCE STARTED, IT CONTINUES TO RUN WITHOUT OUTSIDE INTERFERENCE.
THIRDLY, IT IS ELECTRONIC; THAT IS, IT IS MADE UP OF ELECTRONIC CIRCUITS AND RUNS ON ELECTRICAL ENERGY.
FOURTHLY,THE COMPUTER IS A SYMBOL MANIPULATOR. IT MANIPULATES DATA, NOT PHYSICAL ENTITIES.
FIFTH, THE COMPUTER CAN PERFORM ONLY THE PROCESSES OF ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION, AND COMPARISON, IN ADDITION TO DATA TRANSFER BETWEEN COMPUTERS.
SIXTH, SOMEONE (THE PROGRAMMER) MUST PREPARE A FINITE SEQUENCE OF THE ALLOWABLE INDIVIDUAL OPERATIONS (A PROGRAM) FOR THE COMPUTER TO FOLLOW.
FINALLY, THE COMPUTER CAN STORE THE PROGRAM WITHIN ITS OWN MEMORY AND THEN FOLLOW IT THROUGH UNDER ITS OWN DIRECTION WITHOUT FURTHER GUIDANCE.

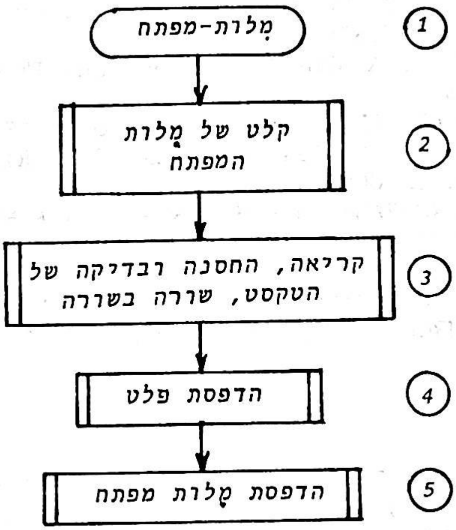
הפלט נראה כך:

```
 1 THE AUTOMATIC ELECTRONIC DIGITAL COMPUTER IS A MACHINE THAT USES
 2 ELECTRONIC CIRCUITS TO MANIPULATE DATA EXPRESSED IN A SYMBOLIC FORM
 3 ACCORDING TO SPECIFIC RULES IN A PREDETERMINED WAY. LET'S EXPLAIN
 4 THIS IN DETAIL.
 5 FIRST OF ALL, THE COMPUTER IS A MACHINE.THIS MEANS IT CAN PERFORM
 6 ONLY THOSE ACTIVITIES FOR WHICH THE BASIC CAPABILITIES HAVE BEEN
 7 SPECIFICALLY DESIGNED INTO THE MACHINE.
 8 SECONDLY, THE COMPUTER IS AUTOMATIC. THIS MEANS THAT ONCE STARTED,
 9 IT CONTINUES TO RUN WITHOUT OUTSIDE INTERFERENCE.
10 THIRDLY, IT IS ELECTRONIC; THAT IS, IT IS MADE UP OF ELECTRONIC
11 CIRCUITS AND RUNS ON ELECTRICAL ENERGY.
12 FOURTHLY,THE COMPUTER IS A SYMBOL MANIPULATOR. IT MANIPULATES DATA,
13 NOT PHYSICAL ENTITIES.
14 FIFTH, THE COMPUTER CAN PERFORM ONLY THE PROCESSES OF ADDITION,
15 SUBTRACTION, MULTIPLICATION, DIVISION, AND COMPARISON, IN ADDITION
16 TO DATA TRANSFER BETWEEN COMPUTERS.
17 SIXTH, SOMEONE (THE PROGRAMMER) MUST PREPARE A FINITE SEQUENCE OF
18 THE ALLOWABLE INDIVIDUAL OPERATIONS (A PROGRAM) FOR THE COMPUTER TO
19 FOLLOW.
20 FINALLY, THE COMPUTER CAN STORE THE PROGRAM WITHIN ITS OWN MEMORY
21 AND THEN FOLLOW IT THROUGH UNDER ITS OWN DIRECTION WITHOUT FURTHER
22 GUIDANCE.

INDEX

COMPUTER, 1, 5, 8, 12, 14, 18, 20
DATA, 2, 12, 16
MACHINE, 1, 5, 7
```

תרשים זרימה של המודולים הבסיסיים של בעיה זו ניתן להלן:



במבנה (4) מדפיסים את שורות התמליל. במבנה (5) מדפיסים רשימה של מלות המפתח כאשר ליד כל אחת מהן מופיעה רשימת השורות בהן היא מופיעה.

על פי תרשים הזרימה אפשר לכתוב את התכנית הבאה:

```
PROGRAM  MILOT_MAFTEACH (INPUT, OUTPUT);
CONST
   ORECH_MILA_MAXIMALI= 20;
   ORECH_SHURA_MAXIMALI= 80;
   MISPAR_MILOT_MAFTEACH_MAXIMALI= 10;
   MISPAR_SHUROT_MAXIMALI= 1000;
TYPE
   SUG_MISPAR_SHURA = 1..MISPAR_SHUROT_MAXIMALI;
   SUG_MISPAR_MILOT_MAFTEACH=1..MISPAR_MILOT_MAFTEACH_MAXIMALI;
   SUG_ORECH_MILA= 1..ORECH_MILA_MAXIMALI;
   SUG_ORECH_SHURA= 1..ORECH_SHURA_MAXIMALI;
   MILA =
         RECORD
            ORECH: SUG_ORECH_MILA;
            EYUT: ARRAY[SUG_ORECH_MILA] OF CHAR;
            HOFAOT: ARRAY[SUG_MISPAR_SHURA] OF BOOLEAN
         END;
   SHURA: PACKED ARRAY[SUG_ORECH_SHURA] OF CHAR;
VAR
   SHUROT: ARRAY[SUG_MISPAR_SHURA] OF SHURA;
   MILAT_MAFTEACH: ARRAY[SUG_MISPAR_MILOT_MAFTEACH] OF MILA;
   TAV: CHAR;
```

```
      I: SUG_ORECH_SHURA;
      ORECH_HAKATAN_BEYOTER: SUG_ORECH_MILA;
      MISPAR_MILOT_MAFTEACH: SUG_MISPAR_MILOT_MAFTEACH;
      ORECH_SHURA: 0..ORECH_SHURA_MAXIMALI;
      MONE_SHUROT: 0..MISPAR_SHUROT_MAXIMALI;

FUNCTION  OTIYOT_GDOLOT (CH: CHAR): CHAR;
    BEGIN
      IF ('a' <= CH) AND (CH <= 'z') THEN
         OTIYOT_GDOLOT:= CHR (ORD(CH)-32)
      ELSE
         OTIYOT_GDOLOT:= CH
    END;

PROCEDURE KRA_MILOT_MAFTEACH;
   VAR
      I: SUG_MISPAR_MILOT_MAFTEACH;
      J: 0..ORECH_MILA_MAXIMALI;
   BEGIN
      ORECH_HAKATAN_BEYOTER:= ORECH_MILA_MAXIMALI;
      FOR   I:= 1 TO MISPAR_MILOT_MAFTEACH DO
         WITH MILAT_MAFTEACH[I]  DO
           BEGIN
              J:=1;
              EYUT[J]:= '^';
              WHILE  NOT  EOLN  DO
                 BEGIN
                  READ (TAV);
                  IF  J < ORECH_MILA_MAXIMALI  THEN
                     BEGIN
                       J:= J+1;
                       EYUT[J]:= OTIYOT_GDOLOT (TAV)
                     END
                 END;
                 ORECH:= J;
                 IF  ORECH < ORECH_HAKATAN_BEYOTER  THEN
                     ORECH_HAKATAN_BEYOTER:= ORECH;
                 READLN
           END
   END;

PROCEDURE MTZA_MILOT_MAFTEACH(MISPAR_SHURA: SUG_MISPAR_SHURA);
    VAR
          K: 0..ORECH_SHURA_MAXIMALI;
          J: 0..ORECH_MILA_MAXIMALI;
          DOMEH, ZEHEH: BOOLEAN;
    BEGIN
      FOR  I:= 1  TO  MISPAR_MILOT_MAFTEACH  DO
        WITH MILAT_MAFTEACH[I] DO
          BEGIN
```

```
            J:= 1;
            K:=0;
            REPEAT
              REPEAT
                J:= J+1;
                DOMEH:=  EYUT[J] = SHUROT[MISPAR_SHURA][J+K-1]
              UNTIL  NOT  DOMEH  OR J= ORECH;
              ZEHEH:= DOMEH AND ((SHUROT[MISPAR_SHURA][J+K] <
                'A')  OR  (SHUROT[MISPAR_SHURA][J+K] > 'Z'));
              K:= K+1;
              J:=0
            UNTIL  ZEHEH  OR (K+ORECH-1 > ORECH_SHURA);
            HOFAOT[MISPAR_SHURA]:= ZEHEH
          END
    END;

PROCEDURE HADPASAT_PELET (NO_LINES: SUG_MISPAR_SHURA);
    VAR
       I: SUG_MISPAR_SHURA;
    BEGIN
      PAGE (OUTPUT);
      FOR  I:= 1  TO  NO_LINES  DO
           BEGIN
             WRITE (I:3, '^':2);
             WRITELN (SHUROT[I])
           END
    END;

PROCEDURE HADPASAT_MILOT_MAFTEACH(NO_LINES: SUG_MISPAR_SHURA;
                       NO_KEYWORDS: SUG_MISPAR_MILOT_MAFTEACH);
            VAR
               I: SUG_MISPAR_MILOT_MAFTEACH;
               J: SUG_ORECH_MILA;
               K: SUG_MISPAR_SHURA;
            BEGIN
               WRITELN;
               WRITELN ('INDEX');
               WRITELN;
               FOR  I:= 1  TO NO_KEYWORDS  DO
                   WITH MILAT_MAFTEACH[I] DO
                       BEGIN
                         FOR  J:= 2  TO  ORECH  DO
                              WRITE EYUT[J];
                         FOR  K:= 1  TO  NO_LINES  DO
                            IF HOFAOT[K]  THEN
                               WRITE (',^', K:1);
                         WRITELN
                       END
            END;
```

```
(117)  PROCEDURE KELET_SHEL_MILOT_MAFTEACH;
(118)    BEGIN
(119)     REPEAT
(120)      WRITELN ('מספר מלות המפתח לתכנית זו יכול להיות');
(121)      WRITELN ('^',MISPAR_MILOT_MAFTEACH_MAXIMALI:2,'בין 1 ל-');
(122)      READLN (MISPAR_MILOT_MAFTEACH)
(123)     UNTIL MISPAR_MILOT_MAFTEACH >= 1  AND
(124)         MISPAR_MILOT_MAFTEACH <= MISPAR_MILOT_MAFTEACH_MAXIMALI;
(125)      WRITELN ('עכשיו הקש את המלים:');
(126)      KRA_MILOT_MAFTEACH
(127)    END;

(128)  PROCEDURE TIPUL_BATAMLIL;
(129)              (* קריאה, אחסון ובדיקה של התמליל, שורה בשורה *)
(130)    BEGIN
(131)      WRITELN ('הכנס את הקטע');
(132)      MONE_SHUROT:=0;
(133)      WHILE NOT EOF AND (MONE_SHUROT<MISPAR_SHUROT_MAXIMALI) DO
(134)        BEGIN
(135)          ORECH_SHURA:= 0;
(136)          MONE_SHUROT:= MONE_SHUROT+1;
(137)          WHILE NOT EOLN DO
(138)           BEGIN
(139)            READ (TAV);
(140)            ORECH_SHURA:= ORECH_SHURA+1;
(141)            SHUROT[MONE_SHUROT][ORECH_SHURA]:=OTIYOT_GDOLOT(TAV)
(142)           END;
(143)          FOR I:= ORECH_SHURA+1  TO ORECH_SHURA_MAXIMALI  DO
(144)              SHUROT[MONE_SHUROT][I]:= '^';
(145)          READLN;
(146)          IF ORECH_SHURA >= ORECH_HAKATAN_BEYOTER-1  THEN
(147)             MTZA_MILOT_MAFTEACH (MONE_SHUROT)
(148)          ELSE
(149)             FOR  I:= 1  TO  MISPAR_MILOT_MAFTEACH  DO
(150)                MILAT_MAFTEACH[I].HOFAOT[MONE_SHURA]:= FALSE
(151)        END
(152)    END;

(153)  BEGIN                  (* התכנית הראשית *)
(154)    KELET_SHEL_MILOT_MAFTEACH;
(155)    TIPUL_BATAMLIL;
(156)    HADPASAT_PELET (MONE_SHURA);
(157)    HADPASAT_MILOT_MAFTEACH (MONE_SHUROT, MISPAR_MILOT_MAFTEACH)
(158)  END.
```

<u>הסבר:</u>

רשימת מלות המפתח מוחזקת במערך של רשומות. המערך מוגדר בשורה (21) והרשומות מתוארות בשורות (12)-(17).

התכנית הראשית בשורות (153) עד (158) מחולקת ל-4 פרוצדורות בסיסיות אשר נסביר אותן להלן:

<u>שורה (154):</u> קריאה לפרוצדורה KELET_SHEL_MILOT_MATEACH, שורות (117) עד (127).

(124)-(122) המשתנה MISPAR_MILOT_MAFTEACH צריך לקבל ערך קביל.

(126) בשורה זו יש קריאה לפרוצדורה KRA_MILOT_MAFTEACH, אשר קוראת מלות מפתח. נסביר עתה פרוצדורה זו, אשר בשורות (35)-(60).

(35) כאשר בודקים שורה כדי לראות אם היא מכילה מלות מפתח, אין צורך לבדוק אם אורכה קטן ממלת המפתח הקטנה ביותר. את אורכה של מלת המפתח הקטנה ביותר מוצאים בזמן הקריאה של מלות המפתח ומחזיקים אותו במשתנה ORECH_HAKATAN_BEYOTER, המוצהר בשורה (24).

(40) נותנים ערך התחלתי מקסימלי למשתנה אשר מחזיק את אורך מלת המפתח. כאשר נקרא מלת מפתח שאורכה קטן מאותו ערך מקסימלי, התוכן של ORECH_HAKATAN_BEYOTER ישתנה בהתאם.

(45) מציבים את התו רווח (^) בתו הראשון המיועד לכל מלת מפתח. אם לא היינו עושים זאת, היה אפשר להגיע למסקנה שמוצאים מלת מפתח גם כאשר היא רק חלק ממלה יותר גדולה. למשל, אם מלת המפתח היא THING אפשר לחשוב שהמלה SOMETHING מתאימה גם כן, מכיון שהיא מכילה את התווים שמחפשים. אבל, כאשר מגדירים את הרווח כחלק ממלת המפתח ^THING, המלה SOMETHING אינה מכילה אותה.

(54)-(46) קוראים את אותיות המלה אחת אחרי חברתה. מניחים שמיד אחרי האות האחרונה של המלה הקישו CARRIAGE RETURN ולכן EOLN מופעל. בשורה (52) מעבירים כל אות שמוזנת למחשב דרך הפונקציה OTIYOT_GDOLOT. תפקידה להפוך אותיות קטנות לגדולות ע"י חיסור של 32 מקוד ASCII שלהן. לולא עשינו טרנספורמציה זו, לא ניתן היה לגלות זהות של שתי מלים אם אחת כתובה באותיות קטנות והאחרת באותיות גדולות.

(55) מעבירים את אורכה של המלה לשדה MILAT_MAFTEACH[I].ORECH.

(57)-(56) מעדכנים את המשתנה ORECH_HAKATAN_BEYOTER.

<u>שורה (155):</u> קריאה לפרוצדורה TIPUL_BATAMLIL, שורות (128) עד (152).

בשלב זה עוברים לפרוצדורה לקריאה, אחסון ובדיקה של התמליל. מטפלים בכל שורה בנפרד בכדי לראות אילו מלות מפתח נמצאות בה. נסביר זאת.

(132) משווים את MONE_SHUROT ל-0 וקוראים את השורות אחת אחרי חברתה, כאשר MONE_SHUROT גדל באחד לפני קריאת כל שורה.

(141) מסיבים את כל האותיות לאותיות גדולות לפני שמעבירים אותן למקומן המתאים במערך הדו-ממדי SHUROT, בשורה MONE_SHUROT ובעמודה ORECH_SHURA.

(144)-(143) ממלאים את שאר העמודות בשורה MONE_SHUROT ברווחים כדי למחוק את התווים שאולי היו שם קודם.

(146) מחפשים מלת מפתח בשורה שעכשיו נקראה, בתנאי שאורכה שווה לפחות לאורכה של מלת המפתח הקטנה ביותר פחות 1. מחסרים 1, כי התו הראשון בכל מלה במערך EYUT היה רווח וזה הגדיל את אורכה ב-1. אם מלת המפתח הקטנה ביותר היא המלה היחידה בשורה, אורכה של שורה זו יהיה ORECH_HAKATAN_BEYOTER-1 ולכן בודקים כל שורה שאורכה שווה או גדול מאורך זה.

<u>שורה 147:</u> קריאה לפרוצדורה MTZA_MILOT_MAFTEACH, שורות (61) עד (84).

(75) בפעם הראשונה משווים את EYUT[2] אל SHUROT[MISPAR_SHUROT][1]. מתחילים במקום השני ב-EYUT כי המקום הראשון הוא רווח, כדי שהמלה לא תהיה צמודה למלה שלפניה. בתחילת השורה המלה איננה יכולה להיות צמודה למלה שלפניה ולכן דרישה זו איננה מתאימה למקרה זה.

מלה בשורה תהיה זהה למלת מפתח אם יש זהות בין האותיות שלה לבין אותיות של מלת המפתח. אבל עדיין יש אפשרות שאין זהות מלאה בין שתי מלים, אם המלה בשורת הקלט יותר ארוכה ממלת המפתח. למשל, אם מלת המפתח היא SOME ובתוך השורה כתוב SOMETHING, המשתנה DOMEH יקבל ערך בוליאני TRUE על אף שאין זהות. לכן בודקים תנאי שני בשורה (77).

(78)-(77) יש לקבוע אם יש זהות מלאה בין שתי המלים או לא. יש זהות רק אם מגיעים למסקנה שיש דמיון (שורה (75)) וגם התו הבא אחרי האות האחרונה איננו אות אלא תו אחר. יש לדאוג לכך שאורך השורות בפועל לא יעלה על ORECH_SHURA_MAXIMALI-1, כי עלולים לנסות להשתמש בשורות (77) ו-(78) במצביע שערכו ORECH_SHURA_MAXIMALI+1, דבר שיביא להפסקת הביצוע. הגבלה זו איננה מורידה מכלליותה של התכנית אם לוקחים זאת בחשבון בזמן קביעת ORECH_SHURA_MAXIMALI.

(79) ממשיכים לחפש מן המקום הבא בשורה.

(80) משווים את J ל-0, כדי להתחיל במחזור הבא של הלולאה עם EYUT[1] (ולא עם EYUT[2] כמו בפעם הראשונה). אחרי ההשוואה הראשונה, חייב להיות רווח משמאל למלה הדומה למלת המפתח, לפני שהיא תחשב כזהה לה לחלוטין.

(81) יוצאים מן הלולאה אם מוצאים בשורה מלת מפתח אחת לפחות כלומר, שהתנאי ליציאה הוא כאשר ZEHEH שווה TRUE. התנאי הנוסף הוא כשאין עוד אפשרות למצוא מלת מפתח בשורה, כי מספר המקומות שנשארו קטן מהאורך של מלת המפתח.

(82) הערך הבוליאני של HOFAOT[MISPAR_SHURA] שווה לערכו של ZEHEH. ערך זה יהיה שווה ל-TRUE אם מוצאים מלת מפתח, אחרת ערכו יהיה שווה ל-FALSE.

<u>שורה (156):</u> קריאה לפרוצדורה HADPASAT_PELET להדפסת התמליל, שורות (85) עד (95).

שורה (157): קריאה לפרוצדורה HADPASAT_MILOT_MAFTEACH להדפסת מלות המפתח, שורות (96) עד (116).

המקום הראשון בכל מלת מפתח, (MILAT_MAFTEACH[I].EYUT[1]), הוא תו רווח ולכן אין מדפיסים אותו שורות (109)-(110). ליד כל מלת מפתח מדפיסים את מספרי השורות בהן היא הופיעה לפחות פעם אחת (שורה 113). כדי לעשות זאת נעזרים במערך MILAT_MAFTEACH[I].HOFAOT[K] שמציין אם מלת מפתח I מופיעה בשורה K, ראה שורה (112).

10.4 מחרוזות (*)

בעיה שכיחה בשטח של עיבוד תמלילים היא למצוא את כל המופעים של מלה מסוימת כדי להחליף אותה במלה אחרת. לפתרון בעיה זו, נסתכל על מלים ולפעמים על שורות כמחרוזות בעלות אורך משתנה ונבנה פונקציות המבצעות פעולות מוגדרות על מחרוזות אלו.

בעזרת רשומות נוכל להגדיר מחרוזת (String) כרשומה המורכבת משני איברים, אשר מוצגת בדרך זו:

```
CONST
        MAX = 80;
TYPE
        MISPAR_TAV = 0..MAX;
        MACHROZET = RECORD
                        ORECH: MISPAR_TAV;
                        TAV: PACKED ARRAY[1..MAX] OF CHAR
                    END;
```

מחרוזת ריקה (Empty String) היא מחרוזת, אשר בה ORECH יהיה שווה ל-0. התוכן האמיתי של האיברים במערך TAV איננו חשוב. על סמך הצהרות אלו נוכל לכתוב את הפונקציות והפרוצדורות המוצגות בהמשך.

בפרוצדורות ובפונקציות המוצגות בהמשך כתבנו את המחרוזות השונות כפרמטרים משתנים רק כאשר היינו צריכים לקבל אותן חזרה עם שינויים. אפשר כמובן לכתוב אותן תמיד כפרמטרים משתנים, כדי למנוע העתקה מחדש בכל פעם שמשתמשים בפונקציה או בפרוצדורה.

10.4.1 קריאה של מחרוזת

בפרוצדורה זו נציג קריאה של מחרוזת תווים אשר מוזנים כקלט מן המסוף.

```
PROCEDURE  KRIA  (VAR M:MACHROZET);
  BEGIN
    WITH  M  DO
        BEGIN
          ORECH:= 0;
          WHILE (ORECH < MAX) AND (NOT EOLN) DO
             BEGIN
                ORECH:= ORECH+1;
                READ  (TAV[ORECH])
             END
        END;
    READLN
  END;
```

10.4.2 הדפסת מחרוזת

בפרוצדורה זו מתבססים על התכונה של <u>מערך ארוז</u> (Packed Array), שמאפשרת לכתוב את המערך M.TAV כולו בפקודת WRITELN אחת. המחרוזת תודפס ביחידת הפלט הסטנדרטית של המערכת.

```
PROCEDURE KTIVA (M: MACHROZET);
  BEGIN
    WRITELN (M.TAV)
  END;
```

10.4.3 חילוץ של תת-מחרוזת

נבצע <u>חילוץ</u> (Extract) של תת-מחרוזת TM מתוך מחרוזת M, ללא שינוי של המחרוזת המקורית. תת-המחרוזת מתחילה מתו מספר HATCHALA ואורכה הוא ORECH_MACHROZET.

```
 (1) PROCEDURE TAT_MACHROZET (M: MACHROZET ;
 (2)                         VAR TM: MACHROZET;
 (3)                         HATCHALA, ORECH_MACHROZET:
 (4)                                      MISPAR_TAV);
 (5)      BEGIN
 (6)        IF HATCHALA + ORECH_MACHROZET - 1 > M.ORECH THEN
 (7)           ORECH_MACHROZET:= M.ORECH - HATCHALA + 1;
 (8)        FOR I:=1  TO  ORECH_MACHROZET DO
 (9)              TM.TAV[I]:= M.TAV[HATCHALA + I - 1];
(10)        TM.ORECH:= ORECH_MACHROZET
(11)      END;
```

<u>הסבר:</u>

(7)-(6) בוחנים אם אורך תת-מחרוזת שמבקשים גדול ממספר התווים שבין מקום ההתחלה לבין סוף המחרוזת. אם זה קורה, מקטינים את האורך של תת-המחרוזת המבוקשת למספר המקומות שבין המקום ההתחלתי (HATCHALA) לבין סוף המחרוזת: M.ORECH - HATCHALA + 1.

(9)-(8) העתקה מן המחרוזת המקורית למחרוזת TM שתחזיק את תת-המחרוזת.

10.4.4 הוספה של מחרוזת למחרוזת קיימת

בפרוצדורה זו מוסיפים מחרוזת MH למחרוזת M החל במקום מוגדר, MAKOM.

```
(1)   PROCEDURE HOSAFA (VAR M: MACHROZET;
(2)                      MH: MACHROZET; MAKOM: MISPAR_TAV);
(3)     VAR
(4)          I,J: MISPAR_TAV;
(5)     BEGIN
(6)        IF  MAKOM <= M.ORECH+1  THEN
(7)          BEGIN
(8)            IF  M.ORECH + MH.ORECH <= MAX  THEN
(9)                M.ORECH:= M.ORECH + MH.ORECH
(10)           ELSE
(11)               M.ORECH:= MAX;
(12)           FOR  I:= M.ORECH  DOWNTO  MAKOM+MH.ORECH DO
(13)               M.TAV[I]:= M.TAV[I-MH.ORECH];
(14)           IF  M.ORECH >= MAKOM + MH.ORECH  THEN
(15)               J:= MAKOM + MH.ORECH -1
(16)           ELSE
(17)               J:= M.ORECH;
(18)           FOR  I:= MAKOM  TO  J  DO
(19)               M.TAV[I]:= MH.TAV[I-MAKOM+1]
(20)        END
(21)    END;
```

הסבר:

(6) תחילה חייבים לבדוק אם התוצאה של הוספת המחרוזת MH למחרוזת הקיימת M תפיק מחרוזת חוקית, ז"א מחרוזת רציפה. אם מקום ההוספה (MAKOM) הוא בתוך המחרוזת M, או מייד אחריה, אין כל בעיה. אבל אם מקום ההוספה הינו מעבר לאורכה של מחרוזת M ועוד 1, יווצר רווח בין סופה של המחרוזת M ומקום ההוספה והפרוצדורה לא תטפל במקרה כזה. אם כוונת המתכנת להוסיף רווחים בין שתי המחרוזות ולהרכיב מהן מחרוזת אחת, אפשר לעשות זאת בשני שלבים: תחילה מוסיפים מחרוזת של רווחים שאורכה MAKOM-M.ORECH ואח"כ מוסיפים את המחרוזת MH.

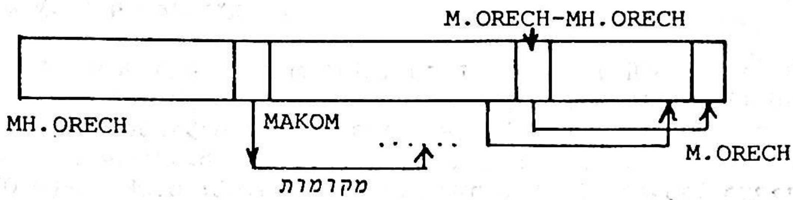
(11)-(8) צריכים לדאוג לכך שהאורך החדש של M לא יהיה גדול מן האורך המקסימלי (MAX) שהוגדר עבור המחרוזת. אם זהו המצב, מבצעים את ההוספה, אבל התווים שהיו צריכים להופיע אחרי MAX לא יוכנסו למחרוזת המשולבת, שורות (12)-(19).

את ההוספה מבצעים בשני שלבים:

<u>שלב א, (13)-(12):</u> העברת התווים ימינה כדי לפנות מקום למחרוזת הנוספת. צריך להעביר כל תו בשיעור של MH.ORECH מקומות ימינה. לכן, תו שהיה במקום I-MH.ORECH יעבור למקום I, שורה (13). צריך להתחיל את ההעברה הזו מן התו שנמצא במקום MAKOM ולהעביר כל תו בשיעור של MH.ORECH

מקומות, עד לתו שנמצא במקום M.ORECH-MH.ORECH. בשורות (9) ו-(11) קבענו את ערכו של המשתנה M.ORECH כאורך של המחרוזת המשולבת.

המצב מתואר בשרטוט הבא:



אם קיים: M.ORECH < MAKOM + MH.ORECH הלולאה לא תתבצע. כלומר, אם האורך של המחרוזת החדשה קטן מן הסכום של מקום ההוספה ואורכה של המחרוזת הנוספת, אזי אין תווים להזיז ימינה.

<u>שלב ב, (19)-(14):</u> מילוי של מחרוזת M החל מ-MAKOM ועד MAKOM+MH.ORECH-1 בתווים של מחרוזת MH. לכן בשורה (18) מאפשרים ל-I להשתנות מ-MAKOM עד MAKOM+MH.ORECH-1 ולקבל את התווים ממקום 1 עד MH.ORECH מן המחרוזת הנוספת. הבעיה היחידה שיכולה להתעורר היא, שאין מקום בשביל המחרוזת הנוספת MH כולה, כי: MAKOM+MH.ORECH > M.ORECH (שורה (17)-(14)). אם זהו המצב, ממלאים את M.TAV[I] עד שקיים I=M.ORECH, על אף שלא מכניסים את כל התווים של המחרוזת הנוספת. פרוצדורה זו מבצעת גם חיבור של שתי מחרוזות, ז"א הוספת מחרוזת מיד אחרי המחרוזת המקורית, כאשר הערך של MAKOM הוא M.ORECH+1.

10.4.5 מחיקה של חלק ממחרוזת קיימת

```
(1)   PROCEDURE HASHMATA (VAR   M: MACHROZET;
(2)                             HATCHALA, ORECH: MISPAR_TAV);
(3)      VAR
(4)         I, GVUL: MISPAR_TAV;
(5)      BEGIN
(6)         GVUL:= HATCHALA+ORECH;
(7)         IF  GVUL > M.ORECH+1  THEN
(8)            GVUL:= M.ORECH+1;
(9)         FOR  I:=0  TO  M.ORECH-GVUL  DO
(10)           M.TAV[HATCHALA+I]:= M.TAV[GVUL+I];
(11)        M.ORECH:= M.ORECH - (GVUL-HATCHALA)
(12)     END;
```

<u>הסבר:</u>

(6) המשתנה GVUL מציין את גבול המחיקה, ז"א את התו הראשון לאחר התווים שעומדים להמחק. תו זה וכל הבאים אחריו, יצטרכו לזוז שמאלה לכיוון ראש המחרוזת ולמלא את הרווח שנוצר ע"י מחיקת התווים. בדרך כלל, התו הראשון

שאיננו למחיקה יהיה התו שנמצא במקום HATCHALA+ORECH, כאשר HATCHALA הוא המקום הראשון למחיקה ו-ORECH הוא מספר התווים למחיקה.

(7) יתכן שמספר התווים למחיקה (ORECH) יהיה גדול ממספר התווים שבמחרוזת המקורית החל ממקום HATCHALA ואילך. במקרה זה נציב את M.ORECH+1 במשתנה GVUL, שמייצג את סוף קבוצת התווים שעומדים למחיקה. התוצאה היא, שנמחק תווים רק עד סוף המחרוזת.

(9)-(10) כאשר מוחקים קטע של מחרוזת, כל תו, החל מ-GVUL ועד M.ORECH, חייב לזוז ORECH מקומות שמאלה. ההזזה היא שמאלה, ממקום GVUL+I למקום HATCHALA+I, עד שמגיעים לתו שמקומו הוא M.ORECH. כאשר מוחקים עד לסוף המחרוזת, האינדקס בפקודה FOR ישתנה מ-0 עד (M.ORECH-GVUL:=M.ORECH-(M.ORECH+1), אשר שווה -1 והגבול העליון יהיה קטן מן הגבול התחתון. במצב זה הלולאה לא תתבצע.

(11) האורך החדש של המחרוזת יהיה שווה לאורך הקודם פחות מספר התווים שנמחקו. במחיקה רגילה מספר זה שווה ל-ORECH. כאשר מוחקים עד סוף המחרוזת, מספר זה שווה ל: M.ORECH-HATCHALA+1. בשני המקרים, מספר התווים למחיקה הוא GVUL-HATCHALA והאורך החדש של המחרוזת הוא: M.ORECH-(GVUL-HATCHALA).

10.4.6 חיפוש מחרוזת מסוימת בתוך מחרוזת אחרת

```
FUNCTION CHIPUS (M, DUGMA: MACHROZET; HATCHALA: MISPAR_TAV):
                                                   INTEGER;
  VAR
       I, J: MISPAR_TAV;
       LO_SHAVEH: BOOLEAN;
  BEGIN
    IF (HATCHALA+DUGMA.ORECH > M.ORECH+1) OR (DUGMA.ORECH=0)
       THEN  CHIPUS:=0
    ELSE
       BEGIN
         I:= HATCHALA-1;
         REPEAT
           I:= I+1;
           J:= 0;
           REPEAT
             J:= J+1;
             LO_SHAVEH:= DUGMA[J] <> M.TAV[I+J-1]
           UNTIL LO_SHAVEH  OR  (J=DUGMA.ORECH)
         UNTIL (NOT LO_SHAVEH) OR (I= M.ORECH-DUGMA.ORECH+1);
         IF  LO_SHAVEH  THEN
            CHIPUS:= 0
         ELSE
            CHIPUS:= I
       END
  END;
```

הסבר:

(1) הפונקציה מחפשת את המופע הראשון של המחרוזת המבוקשת DUGMA במחרוזת M, החל ממקום HATCHALA.

(7) בודקים אם מספר התווים מהמקום ההתחלתי לחיפוש עד לסוף המחרוזת קטן ממספר התווים במחרוזת הדוגמא. אם כך המצב, אין כל סיכוי למצוא אותה במחרוזת המקורית. בודקים גם אם מחרוזת הדוגמא היא מחרוזת ריקה. במקרים אלה וגם במקרה שהדוגמא לא נמצאת במחרוזת M, שורה (21), הערך שהפונקציה מחזירה הוא 0.

(11)-(19) נוהל החיפוש: משווים את שתי המחרוזות זו לזו ע"י השוואה של תו לתו, החל ממקום HATCHALA במחרוזת המקורית.

(18) יוצאים מן הלולאה הפנימית באחד משלושה מקרים:
א. מגלים אי-שיוויון.
ב. מוצאים שיוויון, אבל מגיעים לתו האחרון בדוגמא.
ג. מוצאים שיוויון בין שתי המחרוזות עד לתו האחרון בדוגמא. התו האחרון בדוגמא איננו שווה לתו המקביל במחרוזת. זהו מצב של LO_SHAVEH וגם מצב של J=DUGMA.ORECH.

(19) יוצאים מן הלולאה החיצונית בשני מקרים:
א. כאשר יוצאים מן הלולאה הפנימית במצב של שיוויון.
ב. כאשר מגיעים למצב של אי-שיוויון במצב שנשארים פחות תווים במחרוזת M מאשר יש תווים בדוגמא. המשתנה I (שעולה ביחידות של 1) כבר שווה לערכו של הביטוי M.ORECH-DUGMA.ORECH+1 שהוא המקרה האחרון שמאפשר שיוויון.

אם נמצאים במצב של אי שיוויון כאשר מגיעים לשורה (19) ומצב ב' לא קיים, חוזרים על הלולאה החיצונית עד שמגיעים למצב א' או ב'.

10.4.7 קביעת אורך של מחרוזת

```
FUNCTION ORECH(M.MACHROZET): INTEGER;
    BEGIN
       ORECH:= M.ORECH
    END;
```

10.5 דוגמאות לפעולות במחרוזות

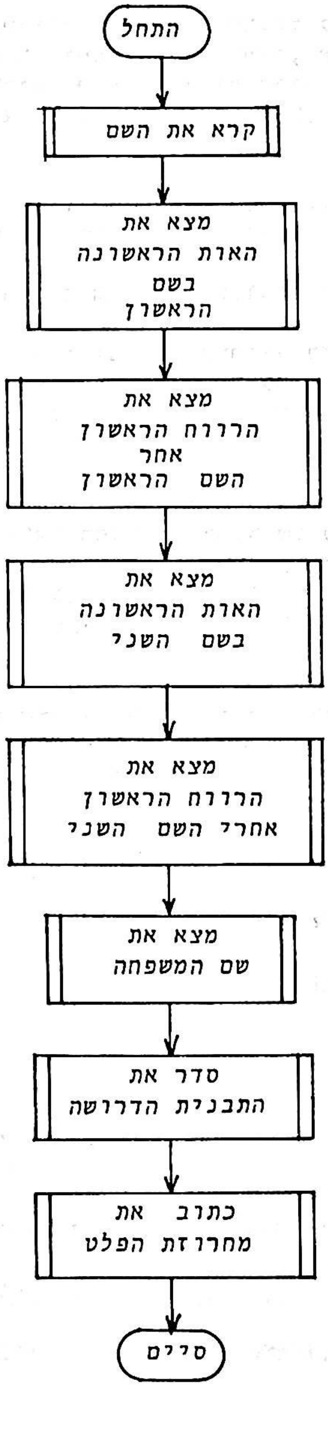
נציג שתי תכניות כדי להדגים את הפעולות השונות שאפשר לבצע במחרוזת.

א. הדפסת שם בראשי תיבות:

נתון שם של אדם (באנגלית): AVRAHAM NIR BLOOM

יש להפיק פלט שיכלול שם משפחה וראשי תיבות של השמות הפרטיים, שיוצגו בצורה זו: BLOOM, A N

השלבים העיקריים של התהליך מתוארים בתרשים זרימה שבהמשך:



התכנית:

```
PROGRAM SIDUR_SHEM (INPUT,OUTPUT);
CONST
     MAX = 80;
TYPE
     MISPAR_TAV= 0..MAX;
     MACHROZET= RECORD
                   ORECH: MISPAR_TAV;
                   TAV: PACKED ARRAY[1..MAX] OF CHAR
                END;
VAR
     SHEM_MISHPACHA, SHEM_RAZUI, PSIK,
     O_R_S_R, O_R_S_S, SHEM, REVACH: MACHROZET;
                 O_R_S_R מתייחס לאות ראשונה, שם ראשון, *)
                 (* O_R_S_S מתייחס לאות ראשונה, שם שני.
     I,J: INTEGER;

                 (* תוכל להניח שכאן מפורטות הפונקציות והפרוצדורות
                 שבסעיפים 10.3.1 עד 10.3.7 *)
PROCEDURE KRIA
PROCEDURE KTIVA
PROCEDURE TAT_MACHROZET
PROCEDURE HOSAFA
PROCEDURE HASHMATA
FUNCTION CHIPUS
FUNCTION ORECH

BEGIN
   REVACH.TAV:= ' ';          (* הגדרת מחרוזת שמכילה רווח אחד *)
   REVACH.ORECH:= 1;
   PSIK.TAV:= ',';            (* הגדרת מחרוזת שמכילה פסיק אחד *)
   PSIK.ORECH:= 1;
   KRIA (SHEM);               (* קריאה של השם המלא *)
   I:= ORECH (SHEM);
HOSAFA (SHEM, REVACH, I+1)     (* הוספה של רווח לאחר השם
      כדי להבטיח ששם המשפחה יהיה מוקף רווחים משני צדדיו. *)
TAT_MACHROZET (SHEM, O_R_S_R, 1, 1);
   (* הוצאת התו הראשון במחרוזת SHEM והכנסתו למחרוזת O_R_S_R,
      כדי לבודד את האות הראשונה בשם הראשון. *)
I:= CHIPUS (SHEM, REVACH, 1);    (* מציאת הרווח הראשון *)
TAT_MACHROZET (SHEM, O_R_S_S, I+1, 1);
   (* הוצאת התו הראשון אחרי הרווח הראשון, שהוא למעשה האות
      הראשונה בשם השני. בזה בודדנו את האות הראשונה בשם השני. *)
I:= CHIPUS (SHEM, REVACH, I+1);
   (* מציאת הרווח השני שמפריד בין השם השני לבין שם המשפחה *)
J:= CHIPUS(SHEM, REVACH, I);
   (* מציאת הרווח השלישי שמסמן סוף שם המשפחה *)
TAT_MACHROZET (SHEM, SHEM_MISHPACHA, I+1, J-1);
                               (* בידוד שם המשפחה *)
```

```
   I:= ORECH (SHEM_MISHPACHA);      (* קביעת האורך של שם-משפחה *)
   HOSAFA (SHEM_MISHPACHA, PSIK, I+1);
                        (* הוספת פסיק בסוף המחרוזת SHEM_MISHPACHA *)
   HOSAFA (SHEM_MISHPACHA, REVACH, I+2);
                                    (* הוספת רווח אחרי הפסיק *)
   HOSAFA (SHEM_MISHPACHA, O_R_S_R, I+3);
                              (* הוספת האות הראשונה של השם הראשון *)
   HOSAFA (SHEM_MISHPACHA, REVACH, I+4);
   HOSAFA (SHEM_MISHPACHA, O_R_S_S, I+5);
                              (* הוספת האות הראשונה של השם השני *)
   SHEM_RAZUI:= SHEM_MISHPACHA;
   KTIVA (SHEM_RAZUI)                (* הצגת השם במבנה הרצוי *)
END;
```

<u>ב. קריאת תמליל והדפסת מלים:</u>

נציג דוגמא נוספת לטיפול במחרוזות בעזרת הפונקציות והפרוצדורות שתוארו קודם. יש לקרוא שורת תמליל (אנגלית) ולהדפיס רשימה של המלים הבודדות המרכיבות אותה.

```
PROGRAM KRIAT_TEXT (INPUT,OUPUT);
CONST
    MAX = 80;
TYPE
    MISPAR_TAV= 0..MAX;
    MACHROZET= RECORD
                   ORECH: MISPAR_TAV;
                   TAV: PACKED ARRAY[1..MAX] OF CHAR
                END;
VAR
   SHURA, MILA, REVACH: MACHROZET;
   I: MISPAR_TAV;

PROCEDURE KRIA          (* תוכל להניח שכאן מפורטות הפונקציות *)
PROCEDURE KTIVA                        (* והפרוצדורות מסעיפים *)
PROCEDURE TAT_MACHROZET                 (* 10.3.1 עד 10.3.7 *)
PROCEDURE HOSAFA
PROCEDURE HASHMATA
FUNCTION CHIPUS
FUNCTION ORECH

BEGIN
      REVACH.TAV[I]:= '^';
      REVACH.ORECH:= 1;
      WHILE NOT EOF DO
          BEGIN
            WRITELN;
            WRITELN ('הדפס שורה בבקשה');
            KRIA(SHURA);
            I:= ORECH(SHURA);
            HOSAFA (SHURA, REVACH, I+1);
```

```
(30)            I:= CHIPUS (SHURA, REVACH, 1);
(31)            WHILE  I > 0  DO
(32)              BEGIN
(33)                TAT_MACHROZET (SHURA, MILA, 1, I-1);
(34)                IF  ORECH(MILA) > 0  THEN
(35)                  BEGIN
(36)                    KTIVA(MILA);
(37)                    WRITELN
(38)                  END;
(39)                HASHMATA (SHURA, 1, I);
(40)                I:= CHIPUS (SHURA, REVACH, 1)
(41)              END
(42)        END
(43) END.
```

<u>הסבר:</u>

קוראים את שורת התמליל בשורה (27) ומוסיפים רווח (29) כדי שהשורה תמיד תסתיים ברווח. מחפשים את הרווח הראשון בשורת התמליל (30). חייב להיות לפחות רווח אחד כי הוספנו תו רווח אחד בסוף השורה. נכנסים ללולאה שמבצעת את הפעולות הבאות:

א. מבודדת את המלה הראשונה, כלומר את התווים עד הרווח הראשון, (שורה (33)).
ב. כותבת מלה זו (שורה (36)).
ג. מוחקת מלה זו מן המחרוזת (שורה (39)).

אם יהיו כמה רווחים יחד, הערך של I בשורה (40) יהיה 1, מכיון שמצאנו רווח במקום הראשון. האורך של MILA בשורה (33) יהיה אז 0 ולא נבצע את השורות (35) עד (38). בשורה (39) נשמיט רווח זה. תהליך זה ימשך עד שנגיע לתו ששונה מרווח.

הערה: בפרק 6 פתרנו תרגיל דומה בלי להיעזר בפרוצדורות, כפי שעשינו כאן.

10.6 תרגילים

10.6.1 שאלות

(1) הצהר על NETUNIM כרשומה עם השדות הבאים:
- שם (מורכב משם משפחה ושם פרטי),
- מספר זהות,
- תאריך לידה (מורכב מחודש, יום ושנה),
- מספר הילדים,
- מצב משפחתי (רווק/ה, נשוי/אה, אלמן/ה, גרוש/ה, נישואים שניים),
- מקום מגורים (כתובת, עיר, מיקוד. הכתובת מורכבת ממספר ושם רחוב),
- מין (זכר, נקבה).

(2) הצהר על משתנה בשם PRATIM מן הסוג NETUNIM המתואר למעלה. הראה איך לקרוא ערכים כדי למלא רשומה זו. עשה זאת עם ובלי שימוש בפקודת WITH וגם על-ידי שימוש בפקודת WITH כפולה כאשר יש צורך.

(3) אילו פקודות והצהרות צריך להוסיף לתכנית MILOT_MAFTEACH כדי שתבצע פעולות נוספות:

א. תספור כמה פעמים מופיעה כל מלה בכל שורה.
ב. תדפיס בסיום את מספר הפעמים שהמלה הופיעה בכל התמליל.

(4) עיין בפרוצדורה להוספת מחרוזת למחרוזת קיימת. שנה אותה לפי ההנחיות הבאות:

א. אם המקום המיועד להוספת המחרוזת MH איננו בתוך המחרוזת המקורית, יש לבצע את ההוספה המבוקשת ולמלא ברווחים את המקומות בין סוף המחרוזת המקורית M והתחלת המחרוזת הנוספת MH.

ב. הפרוצדורה תחזיר את הקוד הבא בסיום:
0 - הוספה רגילה.
1 - הוספת רווחים.
2 - חלק מן המחרוזת הסופית קוצץ.
3 - המקרים (1) ו-(2) ארעו.

(5) בשאלה זו שני חלקים:

א. כתוב פרוצדורה לחיבור של שתי מחרוזות ע"י הוספה של מחרוזת שנייה בסופה של מחרוזת ראשונה. הגדר מחרוזת כרשומה וקרא לפרוצדורה CHIBUR.

ב. השתמש בפרוצדורה CHIBUR, במקום הפרוצדורה HOSAFA, בדוגמאות SIDUR_SHEM ו-KRIAT_TEXT המופיעות בסוף הפרק.

(6) כתוב פרוצדורה אשר מתוארת בעזרת ההצהרה הבאה:

```
PROCEDURE HOTZAA (VAR M, ML: MACHROZET; HATCHALA, ORECH: MISPAR TAV)
```

פרוצדורה זו פועלת כמו HASHMATA, אלא שהיא גם שומרת את המחרוזת שמוחקים במחרוזת חדשה ML.

(7) בשאלה זו שני חלקים:

א. כתוב פרוצדורה המתוארת בעזרת ההצהרה הבאה:

```
PROCEDURE HACHLAFA (VAR  M: MACHROZET, HACHLAFA,
                         DUGMA: MACHROZET;
                         HATCHALA, MEZIA: INTEGER)
```

פרוצדורה זו דומה לפונקציה CHIPUS, בכך שהיא מוצאת את הדוגמא ומעבירה את המקום שבו היא נמצאה למשתנה MEZIA. אם הדוגמא לא נמצאה במחרוזת, MEZIA מקבל ערך 0. בנוסף, פרוצדורה זו מחליפה את המחרוזת DUGMA במחרוזת HACHLAFA. ההחלפה לא תתבצע אם אין מספיק מקום במחרוזת M ויוחזר ערך 1- ב-MEZIA.

ב. הוסף משתנה בוליאני, שיתן למשתמש אופציה להמשיך בהחלפות עד לסוף המחרוזת, אם הוא רוצה בכך.

10.6.2 תכניות

(1) בכיתת סטודנטים יש בין 10 ל-40 תלמידים. נתונים הפרטים האישיים הבאים לכל סטודנט:

- שם פרטי — עד 10 תווים,
- שם משפחה — עד 15 תווים,
- מספר ילדים במשפחה — מ-0 עד 10,
- שנות לימוד של האב — מ-0 עד 20,
- שנות לימוד של האם — מ-0 עד 20,
- ציון ממוצע — מ-0 עד 100.

יש לקבוע אם קיים קשר בין גודל המשפחה לבין סכום שנות הלימוד של ההורים:

א. כתוב תכנית שתוציא כפלט: את השמות של 5 סטודנטים מהמשפחות הגדולות ביותר, השמות של 5 סטודנטים מהמשפחות הקטנות ביותר ואת סכום שנות הלימוד של ההורים של כל סטודנט שנבחר באחת מהקבוצות.
ב. הוצא כפלט את הציון הממוצע בשתי הקבוצות שהוגדרו בשאלה א'.
ג. לכמה סטודנטים יש ממוצע מעל ל-90 ולפחות הורה אחד שלמד פחות מ-10 שנים ?
ד. לכמה סטודנטים יש ממוצע מעל ל-90 ולפחות הורה אחד שלמד יותר מ-10 שנים ?

<u>נתונים</u>

שם		מספר ילדים	שנות לימוד		ציון
פרטי	משפחה		אב	אם	
AVRAHAM	FEDER	7	7	10	81
CHANOCH	COHEN	2	15	14	94
SARA	FRIEDMAN	5	6	8	65
SHALOM	ADIDA	4	10	11	88
CHAVA	COHEN	3	14	12	61
YEHUDIT	PELER	1	16	16	98
SHALOM	AMIR	2	18	14	85
AVRAHAM	MIZRAHI	9	8	13	92
YAKOV	HANOVER	2	9	7	68
DANIEL	LEVI	6	7	8	82
DANIELA	RUBIN	4	10	9	77
ESTHER	HURVITZ	5	13	14	92

(2) נתונים שני מערכים של רשומות:

א. מערך של רשומות סטונדטים.
ב. מערך של רשומות קורסים, כאשר בכל רשומה נמצאים שמות הסטודנטים שלמדו באותו קורס והציון שכל אחד קיבל.

כתוב תכנית שתקלוט מרשומות הקורסים את הציון שכל סטודנט קיבל בקורס שלמד בו ותרשום זאת ברשומה של אותו סטודנט. בסיום הרצת התכנית תודפסנה כל הרשומות של הסטודנטים.

<u>תיאור רשומת סטודנט:</u>

שם:	20 תווים,
מספר סטודנט:	מספר שלם בן 6 ספרות,
כתובת:	שם רחוב ומס' בית, עיר, מיקוד,
דו"ח ציונים:	מספר הקורסים שהתלמיד השתתף בהם, שם של כל קורס והציון שהתלמיד קיבל באותו קורס.

<u>תיאור רשומת קורס:</u>

שם הקורס: 20 תווים.
מספר סטודנטים בקורס: מספר שלם קטן מ-50,
שמות הסטודנטים והציון של כל אחד מהם.

<u>תיאור הפלט:</u>
דוגמא של הדוח שהתכנית תפיק:

שם: יעקב מאיר
מספר סטודנט: 545
כתובת: הרקפת 17
ירושלים 51246

שם הקורס	ציון
ביולוגיה	85
מתמטיקה	46
כימיה	91
אנגלית	82

(3) במערכת הנהלת חשבונות מנהלים שתי קבוצות של נתונים לכל חשבון:

א. נתונים מזהים של כל חשבון:
- שם הלקוח (עד 15 תווים).
- מספר החשבון של אותו לקוח.
- גובה האשראי שמוכנים לאפשר לו.
- המצב הנוכחי של החשבון, אשר יכול להיות בזכות או בחובה.

ברשומה האחרונה שם הלקוח הוא 15 תווים 'X'.

ב. תנועות, שהן פעולות כספיות שהתבצעו במשך החודש האחרון. תנועה מורכבת משני נתונים:

▪ מספר חשבון.
▪ סכום, עם סימן + או עם סימן -.
ברשומה האחרונה מספר החשבון הוא 1-.

התכנית צריכה לעדכן את החשבון של כל בעל חשבון על פי הפעולות שהתבצעו במשך החודש ולהדפיס דוח כפי שמוצג בדוגמא.

קלט לדוגמא:

	AVRAHAMI	14	30000	1731
	YITZCHAKI	16	16000	14221
חשבונות	:			
	:			
	XXXXXXXXX	0	0	0

	16	-12000
תנועות	14	12000
	:	
	-1	

פלט לדוגמא:

לקוח	מספר החשבון	אשראי	יתרה
AVRAHAMI	14	30000	13731
YITZCHAKI	16	16000	2221

(4) בספריה מנהלים את מלאי הספרים. רשומת ספר כוללת את הפרטים הבאים:
▪ כותרת כמחרוזת של תווים.
▪ מחברים (מספר המחברים ושמותיהם).
▪ שם המו"ל.
▪ שנת הוצאה לאור.
▪ מספר סידורי.

כתוב תכנית שקוראת עד 20 רשומות ספרים, ממיינת אותם בסדר עולה לפי שנת ההוצאה לאור ומדפיסה את כל הפרטים של כל ספר בספריה.

(5) כתוב תכנית כדי לקרוא תמליל ולהפריד בין המשפטים השונים. משפט יכול להסתיים בנקודה, בסימן קריאה, או בסימן שאלה שאחריהם יבוא תו רווח. תוכל להיעזר בפרוצדורות לטיפול במחרוזת.

פרק 11
קבצים

11.1 שילוב קבצים בתכנית

קובץ (File) הוא סדרה של איברים מאותו סוג. הקובץ מוחסן ביחידות החסנה היקפיות (Peripheral Storage Units), כמו למשל סרט מגנטי (Tape), דיסק (Disk), או תקליטון (Diskette). אמצעי הרישום של הנתונים נקרא גם מצע החסנה (Media). איברים של מערך, שלא כאיברי הקובץ, מוחסנים בזיכרון המחשב.

השוני שבין צורת ההחסנה של מערך ושל קובץ גורם להבדלים אלה בתכניות:

א. מערך מוחסן בזיכרון ולכן ההצהרה על מערך כוללת את גודלו כדי שהמהדר יקצה לו את המקום הדרוש. קובץ לא מוחסן בזיכרון אלא על סרט או על דיסק שקיבולתם אינה מוגבלת. הזיכרון משמש לו רק "תחנת מעבר" בזמן קריאה לשם עיבוד, או בזמן כתיבה לאחר עיבוד ולפני כתיבה על אמצעי ההחסנה. גודל השטח בזיכרון שמשמש למעבר הנתונים מהקובץ הוא כאורכו של איבר אחד בקובץ. השטח בזיכרון שמחזיק את האיבר נקרא חלון (Window), או מאגר (Buffer) ותוכנו נקרא משתנה המאגר (Buffer Variable).

ב. כל איברי המערך נמצאים תמיד בתוך הזיכרון ולכן אפשר לשלוף כל אחד מהם באופן מיידי ע"י פנייה למקומו היחסי בתוך המערך. אם כותבים למשל OVDIM[37] הכוונה היא לקבל את התוכן של איבר מספר 37 במערך OVDIM שהוזכר בתכנית MASKORET13. לעומת זאת, אפשר לגשת באופן מיידי רק לאיבר של הקובץ אשר נמצא עכשיו במאגר ולא לכל הנתונים של הקובץ. אם מחפשים איבר מסוים, צריך לקרוא את הקובץ מתחילתו ולבדוק כל איבר בנפרד בזמן שהוא נמצא במאגר, או אחרי שמעבירים אותו לתוך משתנה חדש המוגדר לצורך זה.

בפסקל סטנדרטי צריך להצהיר על כל קובץ חיצוני (External File) בהצהרת התכנית - ...PROGRAM. קובץ מוגדר "חיצוני" אם הוא היה קיים לפני ביצוע התכנית או אם יוצרים אותו במהלכה ורוצים שלאחר הביצוע הוא יישמר במערכת. קובץ פנימי (Internal File) יוצרים במהלך התכנית לשם שמירת נתונים באופן זמני וכאשר לא רוצים לשמור אותם בגמר הביצוע. קובץ כזה לא חייב להופיע ברשימת הקבצים שבהצהרת התכנית. יש מהדרים שלא מתייחסים לשמות הקבצים

שבהצהרת התכנית ולכן הקבצים שמוזכרים בה משמשים לתיעוד בלבד.

קובץ נמצא במצע החסנה חיצוני, מחוץ לזיכרון המחשב. כדי שהתכנית תוכל לפעול עם קובץ, יש להגדיר אותו וליצור קשר בינה לבין הקובץ ובסיום העיבוד יש לנתק קשר זה. את יצירת הקשר וניתוקו מבצעים בפקודות פתיחה וסגירה.

בזמן פתיחה של קובץ (File Open) מבצעים את הפעולות הבאות:

א. כתיבת תוית (Label) בקובץ פלט כדי שאפשר יהיה לזהות אותו בעת קריאה.
ב. קריאת התוית של קובץ קלט כדי להבטיח שקוראים את הקובץ הנכון.
ג. קריאת הרשומה הראשונה של קובץ קלט לתוך המאגר.

חייבים לתת הוראה כדי לפתוח כל קובץ קלט וכל קובץ פלט. קובץ הקלט הסטנדרטי, שהוא המקלדת או קורא הכרטיסים וקובץ הפלט הסטנדרטי, מסוף או מדפסת מוגדרים בדרך כלל למהדר. הוא פותח אותם באופן אוטומטי ללא צורך בכתיבת הוראות מפורשות של המתכנת. בתכניות קודמות קראנו וכתבנו ביחידות קלט ופלט סטנדרטיות בלבד ולכן לא עסקנו בפתיחת קבצים.

סגירה של קובץ (File Close) כוללת כתיבה של תוית-סוף-קובץ בעת כתיבה ובדיקה של תוית-סוף-קובץ בעת קריאה. לאחר סגירת הקובץ מנותק הקשר הישיר בינו לבין התכנית ומבוטלים שדות עזר המשמשים להעברת נתונים. סגירה של קבצי קלט ושל קבצי פלט סטנדרטיים נעשית באופן אוטומטי. ישנם מהדרים שבהם גם סגירת קבצים אחרים היא אוטומטית ולכן אין צורך בהוראות מפורשות של המתכנת.

הפעולות השונות לפתיחה ולסגירה של קבצים אינן באחריות המתכנת. עליו רק לרשום את ההוראה המתאימה, אשר תבוצע ע"י פרוצדורה סטנדרטית של המהדר, אשר אינה זהה בכל המהדרים ובכל המחשבים. על הקורא לעיין בספרות הרלוונטית.

הפרוצדורה לפתיחת קבצים מוצגת כאן לשם הסבר בלבד, וכוללת שלבי הפעולה אחדים, לפי הגדרת הקובץ ומצע ההחסנה שלו. שים לב לכך שיכולים להיות הבדלים בין מחשב למחשב.

סרט מגנטי:
קלט: קוראים את הסרט עד אשר נקראת התוית של הקובץ המבוקש. מכאן ואילך מתחילים בקריאה ובעיבוד של הרשומות.
פלט: קוראים את הסרט עד לאחר קריאת תוית הסיום של הקובץ האחרון. כותבים את התוית של הקובץ החדש וממשיכים בכתיבה של רשומות הקובץ.

דיסק מגנטי או תקליטון:
הדיסק והתקליטון הם מצעי החסנה לגישה ישירה ולכן הנוהל לפתיחה ולסגירה של קבצים שונה מאשר בסרט המיועד לגישה סדרתית.
קלט: קוראים את אינדקס הקבצים, אשר מכיל את התויות, מאתרים את שם הקובץ הדרוש ופונים אליו לפי כתובתו לשם קריאה.
פלט: מוסיפים לאינדקס תוית של הקובץ החדש ואת המקום הפנוי שבו אפשר לכתוב אותו במצע ההחסנה. פונים לאותו מקום בדיסק כדי לכתוב את רשומות הפלט.

הפרוצדורה לסגירת קבצים גורמת לקריאה או לכתיבה, בהתאמה, של "תוית סוף קובץ" ולאחריה:-

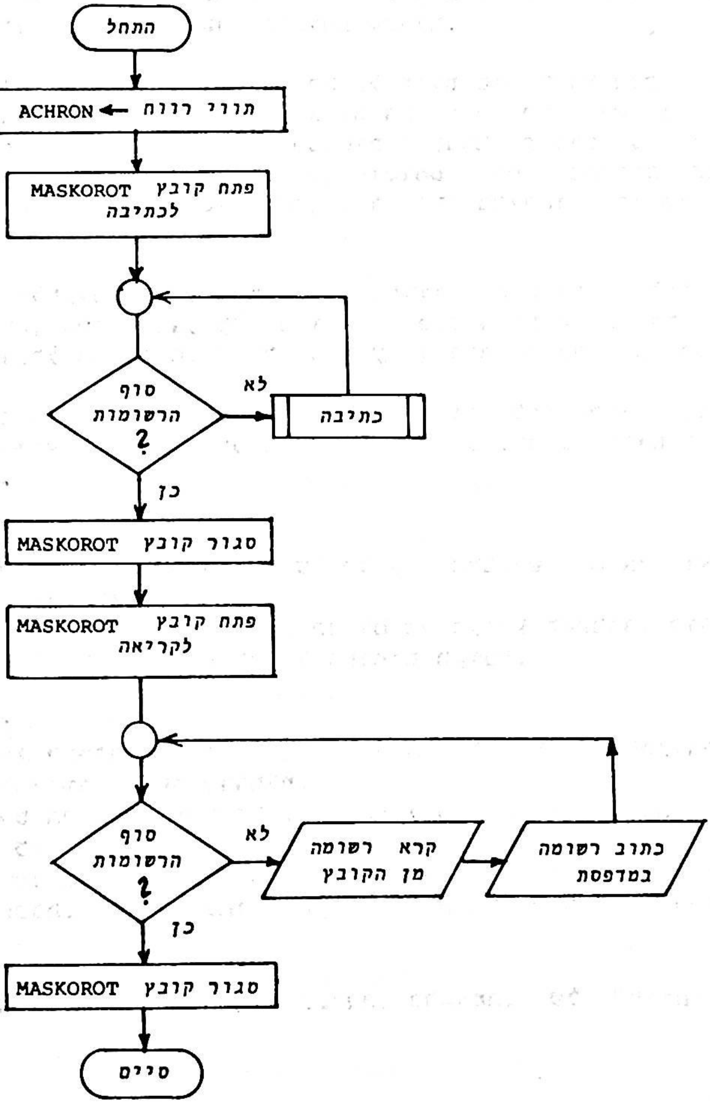
סרט מגנטי: גלגול הסרט לתחילתו.
דיסק או תקליטון: הפרוצדורה חוזרת לאינדקס הקבצים שבמצע ההחסנה וכותבת בו את גודלו של הקובץ החדש. היא תשתמש בנתון זה בעת קריאה של הקובץ הזה וגם כדי לנהל את שטחי ההחסנה הפנויים.

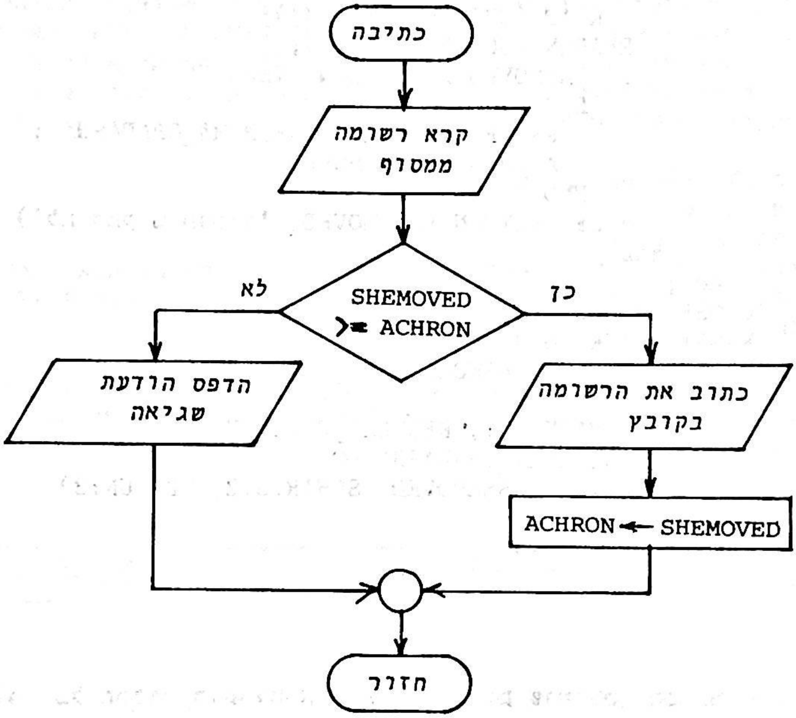
11.1.1 הצהרות ופקודות עיבוד של קבצים

נציג דוגמת תכנות כדי להסביר את ההצהרות ואת פקודות העיבוד השונות:

התכנית הבאה מקבלת רשומות עובדים כקלט ומניחה שהן ממויינות. אחר כך היא כותבת אותן לקובץ שמוחסן בדיסק ומדפיסה אותן. אם מספר סידורי של רשומה כלשהי חורג מן הסדר הנכון, היא לא נכתבת בקובץ ומודפסת הודעת שגיאה.

נתבונן בתרשים זרימה של הבעיה, שבו מוצגות פתיחה וסגירה של קבצים:





על פי התרשים אפשר לכתוב את התכנית שלהלן:

```
PROGRAM MASKORET17 (INPUT, OUTPUT, MASKOROT);
CONST ORECH= 25;
TYPE SHEM= PACKED ARRAY[1..ORECH] OF CHAR;
     RESHUMAT_OVED= RECORD
                        SHEMOVED: SHEM;
                        SCHAR: REAL;
                        TZIYUN: CHAR
                    END;
VAR
     RESHUMA_CHADASHA: RESHUMAT_OVED;
     MASKOROT: FILE OF RESHUMAT_OVED;
     ACHRON: SHEM;
     MAKOM: INTEGER;
BEGIN
   ACHRON= '^^^^^^^^^^^^^^^^^^^^^^^^^';
   REWRITE (MASKOROT);
   WHILE NOT EOF DO
     BEGIN
       WRITELN ('הקש שם, שכר, ציון');
       WITH RESHUMA CHADASHA DO
         BEGIN
           FOR MAKOM := 1 TO ORECH DO
```

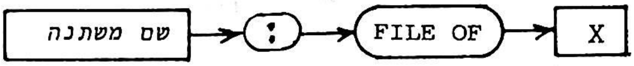
```
(23)              READ (SHEMOVED[MAKOM]);
(24)            READLN (SCHAR, TZIYUN);
(25)            IF SHEMOVED >= ACHRON  THEN
(26)               BEGIN
(27)                 WRITE (MASKOROT, RESHUMA_CHADASHA);
(28)                 ACHRON := SHEMOVED
(39)               END
(30)            ELSE WRITELN (SHEMOVED, 'לא במקום הנכון')
(31)          END
(32)      END;
(33)    CLOSE (MASKOROT);
(34)    RESET (MASKOROT);
(35)    WHILE NOT EOF (MASKOROT) DO
(36)      BEGIN
(37)        READ (MASKOROT, RESHUMA_CHADASHA);
(39)        WITH RESHUMA_CHADASHA DO
(40)           WRITELN (SHEMOVED, SCHAR:8:2, TZIYUN:3)
(41)      END;
(42)    CLOSE (MASKOROT)
(43) END.
```

<u>הסבר:</u>

(1) בתחילת כל תכנית כותבים את שמות הקבצים שנשתמש בהם במהלכה. באמצעות לוח המקשים, קוראת כרטיסים, מדפסת או מסך מזינים ומפיקים קבצים ולכן מזכירים את שמם בהצהרת התכנית: INPUT ו-OUTPUT בהתאמה.

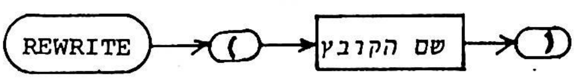
בנוסף לקבצים אלה, שהם <u>הקבצים הסטנדרטיים</u>, בונים קובץ על דיסק שיקרא MASKOROT. בקובץ זה נכתוב רשומות ונקרא אותן בעת הצורך, לשם עיבוד ולשם הצגה במסך או הדפסה. קובץ זה מוגדר כקובץ חיצוני.

(11) מצהירים על משתנה כקובץ בצורה הבאה:



התו X מסמן כל סוג של נתון, למשל מספר שלם או ממשי, תו, או מערך, אבל במקרה השכיח הוא מייצג רשומה. במקרה זה איברי הקובץ הם רשומות, שסוגן מוצהר בשורות (4)-(8).

(16) הפרוצדורה הסטנדרטית לפתיחת קובץ לכתיבה היא:

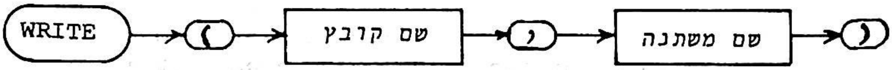


במחשבים מסוימים כותבים את שורה (16) בצורה שחורגת מהנוהל הסטנדרטי. למשל, במחשב PDP כותבים REWRITE (MASKOROT,'MAS.DAT'), כאשר MASKOROT הוא השם הפנימי של הקובץ בתוך התכנית ו-MAS.DAT ממלא את דרישות המערכת לכתיבת שם-קובץ המוחזק בדיסק. הקשר הזה חייב להעשות בכל מחשב, אבל לפעמים הוא נעשה מחוץ לתכנית באמצעות הוראות בקרה של מערכת ההפעלה.

הפרוצדורה REWRITE מבצעת את התפקידים הבאים:

א. מוחקת את הקובץ.
ב. מכניסה לתוך המאגר איבר ריק.
ג. מכוונת את המצביע אל המקום בדיסק שבו יכתב האיבר הבא. ההצבעה היא על סוף הקובץ ולכן הפונקציה הבוליאנית "EOF(קובץ)" תהיה TRUE תמיד בזמן כתיבה.

(27) פרוצדורה סטנדרטית זו (שקיימת רק במהדרים חדשים) מעבירה את התוכן של הרשומה RESHUMA_CHADASHA לקובץ שנקרא MASKOROT. שם הרשומה צריך להיות מן הסוג של איברי הקובץ. הצורה הכללית של פקודת הפרוצדורה היא:



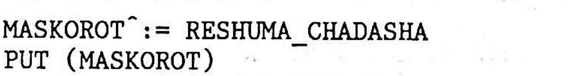
במקום פרוצדורה זו, אפשר לכתוב את הפקודות הבאות שיוסברו בהמשך:

```
MASKOROT^:= RESHUMA_CHADASHA
PUT (MASKOROT)
```

הצירוף של שם הקובץ וסימן החץ (^), כאן MASKOROT^, מתייחס למאגר שממנו מעבירים את איברי הקובץ אל הדיסק. הפקודה הראשונה מכניסה למאגר את התוכן של RESHUMA_CHADASHA. הפרוצדורה הסטנדרטית "PUT(קובץ)" כותבת את המאגר כאיבר חדש בקובץ, מזיזה את המצביע לסוף האיבר החדש, מוחקת את תוכן המאגר ומשאירה את משתנה המאגר בלתי מוגדר. אם ננסה לבצע PUT(MASKOROT) מיד, בלי למלא את המאגר מחדש, תיגרם שגיאת ביצוע כי ניסינו לכתוב מאגר ריק.

(33) סגירת הקובץ. הוראה זו אינה דרושה במהדרים מסוימים, אם הם מבצעים אותה באופן אוטומטי בסיום התכנית.

(34) פתיחת קובץ לקריאה נעשית על ידי הפרוצדורה שהמבנה שלה הוא:



במחשבים מסוימים מוסיפים את השם החיצוני של הקובץ, למשל, במחשב PDP פקודה זו נראית כך: RESET(MASKOROT, 'MAS.DAT')

הפרוצדורה RESET מבצעת 3 פעולות:

א. מכניסה למאגר ^MASKOROT את האיבר הראשון של הקובץ.

ב. מכוונת את המצביע למקום הבא בקובץ.

ג. אם הקובץ לא היה ריק, הפונקציה הבוליאנית "EOF(קובץ)" מקבלת ערך FALSE. אם הקובץ היה ריק, התוכן של המאגר בלתי מוגדר וערכה של הפונקציה הבוליאנית הופך ל-TRUE.

(37) פרוצדורה סטנדרטית זו מעבירה את התוכן של האיבר הבא לתוך RESHUMA_CHADASHA וממלאת את המאגר עם האיבר הבא בקובץ.
במקום פרוצדורה זו ניתן לכתוב:

```
RESHUMA_CHADASHA:= MASKOROT^
GET (MASKOROT)
```

הפקודה הראשונה מעבירה את האיבר הבא בקובץ למשתנה RESHUMA_CHADASHA. הפקודה "GET(קובץ)" מעבירה איבר חדש לתוך המאגר ומקדמת את המצביע לאיבר הבא של הקובץ. אם אין איברים נוספים בקובץ, הפרוצדורה הופכת את הפונקציה הבוליאנית "EOF(קובץ)" מ-FALSE ל-TRUE וגורמת למשתנה המאגר להיות בלתי מוגדר. אם מנסים לבצע שוב פקודת "GET(קובץ)" תגרם שגיאת ביצוע, כי "EOF(קובץ)" שווה ל-TRUE.

במהדרים מסוימים של פסקל, הפונקציה "EOF(קובץ)" אינה מופעלת מיד לאחר קריאת סוף הקובץ. במקרה כזה רצוי להגדיר קבוע בשם DUMMY, שערכו 'ZZZZZZZZZZZZZ' ולכתוב אותו בסוף הקובץ כדי שישמש כדגל. המתכנת יבדוק כל איבר אם הוא שווה ל"דגל" כדי לדעת אם הגיע לסוף הקובץ.
שורה (35) תכתב כך:

```
WHILE (RESHUMA_CHADASHA. SHEMOVED <> DUMMY) DO
```

נסכם עתה את התהליך של כתיבה וקריאה של קבצים כדלהלן:
(נקרא לשם הקובץ KOVETZ ולשם הרשומה RESHUMA).

<u>א. כתיבה (יצירה) של קבצים:</u>

```
(1)  REWRITE (KOVETZ);
(2)  WHILE (תנאי להמשך יצירת רשומות) DO
(3)  BEGIN
(4)    יצירת רשומה
(5)    בדוק את הרשומה החדשה כדי לראות אם היא קבילה
(6)    WRITE (KOVETZ, RESHUMA);
(7)    עדכן את המשתנים לצורך בדיקת התנאים בשורות (2) ו-(5)
(8)  END;
```

<u>ב. קריאה של קבצים:</u>

```
(1)  RESET (KOVETZ);
(2)  WHILE  NOT EOF (KOVETZ) DO
(3)    BEGIN
(4)      READ (KOVETZ, RESHUMA);
(5)        עבד את הרשומה כנדרש
(6)    END;
```

כדי לעבד רשומה צריכים להכניס אותה לזיכרון.
כזכור, ההוראה READ (KOVETZ, RESHUMA) מורכבת משתי פקודות:

```
(a)   RESHUMA := ^KOVETZ
(b)   GET (KOVETZ)
```

רק הפקודה הראשונה נחוצה לפני העיבוד. לכן, כאשר לא משתמשים ב-READ אלא בפקודות (a) ו-(b), כותבים לפעמים את הפקודה (b) אחרי שורה (5) בקטע הקודם (ב).

11.2 קריאה ועיבוד של נתונים

נציג בהמשך דוגמאות של שתי פעולות שבהן משתמשים בהוראת קריאה:

א. <u>בחירה</u> (Selection): כאשר רוצים לבחור מתוך הקובץ אחת או יותר רשומות שיש להן תכונה מסוימת. התכונה יכולה להיות תכונה עצמית, כמו ערך מסוים בשדה כלשהו, או ערך בהשוואה לרשומות אחרות בקובץ, כמו למשל מציאת הרשומה שמכילה שדה עם ערך מינימלי או מקסימלי.

ב. <u>סיכום</u> (Summary): כאשר רוצים לסכם את הערכים אשר בשדה כלשהו בכל רשומות הקובץ לשם בניית טבלה, או לשם חישוב סטטיסטי, כמו למשל חישוב הממוצע.

במקרה הראשון, אם מחפשים רשומה מסוימת מבין כל הרשומות, אפשר להפסיק את החיפוש כאשר מוצאים אותה. במקרה השני, בודקים או מסכמים מאפיין של כל הרשומות בקובץ ולכן חייבים תמיד לסרוק את כולו.

11.2.1 בחירת רשומות מתוך קובץ

הקובץ KRIOT מורכב מרשומות שמכילות נתונים שנאספו במשך תקופה מסוימת. שדות הרשומה הם: מידת החום בירושלים, זמן המדידה ותאריך המדידה. מבנה של רשומה טיפוסית ניתן להלן:

מידת החום	שעה	תאריך
3	1400	13/2/86

התכנית מחשבת את מידת החום המינימלית והמקסימלית על פי הרשומות בקובץ ויוצרת קובץ חדש שמחזיק רק את מידות החום בלי תאריכים וזמנים.

```
 (1)  PROGRAM MIDAT_CHOM (KRIOT, OUTPUT);
 (2)  TYPE
 (3)       KRIA = RECORD
 (4)                MIDAT_HACHOM, ZMAN : INTEGER;
 (5)                TARICH : PACKED ARRAY  [1...8] OF CHAR
 (6)              END;
 (7)  VAR
 (8)     KRIOT : FILE OF KRIA;
 (9)     MIDOT : FILE OF INTEGER;
(10)     MAX, MIN, AVODA: KRIA;
(11)  BEGIN
(12)    RESET (KRIOT);
```

```
(13)    REWRITE (MIDOT);
(14)    READ (KRIOT, AVODA);
(15)    MAX:= AVODA;
(16)    MIN:= AVODA;
(17)    WRITE (MIDOT, AVODA.MIDAT_HACHOM);
(18)    WHILE NOT EOF (KRIOT)  DO
(19)       BEGIN
(20)         READ (KRIOT, AVODA);
(21)         IF AVODA.MIDAT_HACHOM > MAX.MIDAT_HACHOM THEN
(22)           MAX:= AVODA
(23)         ELSE
(24)           IF AVODA.MIDAT_HACHOM < MIN.MIDAT_HACHOM THEN
(25)             MIN:= AVODA;
(26)         WRITE (MIDOT, AVODA.MIDAT_HACHOM)
(27)       END;
(28)    WITH MAX DO
(29)      BEGIN
(30)        WRITELN (MIDAT_HACHOM, '^^מידת החום המקסימלית היתה');
(31)        WRITELN (TARICH, '^^בתאריך', 0.01*ZMAN:5:2, '^^בזמן')
(32)      END
(33)    WITH MIN DO
(34)      BEGIN
(35)        WRITLEN (MIDAT_HACHOM, '^^מידת החום המינימלית היתה');
(36)        WRITELN (TARICH, '^^בתאריך', 0.01*ZMAN:5:2, '^^בזמן')
(37)      END
(38)    CLOSE (KRIOT);
(39)    CLOSE (MIDOT)
(40)  END.
```

<u>הסבר:</u>

(9) הקובץ MIDOT מורכב מאיברים אשר מכילים מספרים שלמים בלבד.

(14)-(16) את הקריאה הראשונה בסריקה זו עושים מחוץ ללולאה. כך קובעים את הקריטריון ההתחלתי שעליו מבססים את האומדן עבור מידת החום המינימלית והמקסימלית.

(17) כותבים את מידת החום בקריאה הראשונה בקובץ MIDOT.

(21)-(25) את הרשומות MAX ו-MIN מחליפים בכל פעם שנתקלים במידת חום אשר גדולה או קטנה, בהתאמה, ממה שהיה בעבר.

(26) את מידת החום כותבים בקובץ MIDOT בכל מקרה.

<u>11.2.2 קריאה, סיכום והדפסה של נתונים</u>

נשתמש בדוגמא לפעולות אלו: נתון קובץ של תנועות בפריטים שונים שנמצאים במלאי. יש להדפיס דוח אשר יכלול את השינוי של הכמות במלאי עבור כל פריט.

קובץ התנועות, או הפעולות, ממויין לפי מספר פריט. מבנה הרשומות בקובץ ניתן להלן:

מספר הפריט	סוג הפעולה	כמות

סוג הפעולה הוא אחד משלושה ערכים:

- מכירה - המלאי יורד לפי הכמות שנמכרה,
- קבלת משלוח - המלאי עולה לפי הכמות שהתקבלה,
- החזרת סחורה - המלאי עולה לפי הכמות שהוחזרה.

תכנית זו נכתבת במתכונת של הצורה הכללית של תכניות לקריאת קבצים.

```
PROGRAM MELAI (PEULOT, OUTPUT);
TYPE
   CODE = PACKED ARRAY [1..6] OF CHAR;
   PEULA = RECORD
              ZEHUT_PARIT : CODE;
              SUG_PEULA : (MECHIRA, KABALA, HACHZARA);
              CAMUT : INTEGER
           END;
VAR
   PEULOT : FILE OF PEULA;
   SHINUI : INTEGER;
   PARIT_NOCHCHI : CODE;
BEGIN
   RESET (PEULOT);
   WRITELN ('מספר פריט ^^^^^ שינוי במשך החודש');
   WRITELN;
   WHILE NOT EOF (PEULOT)  DO
      WITH PEULOT^  DO              (* משתנה במאגר = "XXXX^" *)
         BEGIN
            PARIT_NOCHCHI := ZEHUT_PARIT;
            SHINUI := 0;
            WHILE (ZEHUT_PARIT = PARIT_NOCHCHI) AND
                           NOT EOF (PEULOT)  DO
               BEGIN
                  CASE SUG_PEULA OF
                    MECHIRA : SHINUI:= SHINUI - CAMUT;
                    KABALA, HACHZARA : SHINUI:= SHINUI+CAMUT
                  END;
                  GET (PEULOT)
               END;
            WRITELN (SHINUI, '_':12, ZEHUT_PARIT:9)
         END
END.
```

בהרצה טיפוסית נקבל את הפלט הבא:

מספר פריט	שינוי במשך חודש
A14T37	400
B19L16	-18
D18R33	225
F42S18	-93

<u>הסבר:</u>

(23)-(22) משתמשים בשדה שבמאגר PEULOT^.ZEHUT_PARIT, כדי לבדוק אם הרשומה החדשה מתייחסת לפריט חדש. אם כך הדבר, מדפיסים שורת פלט (שורה 31) וחוזרים לשורה (20), שם רושמים את המספר של הפריט החדש במשתנה PARIT_NOCHCHI ונכנסים שוב פעם ללולאה. אם הרשומה הבאה שייכת לאותו פריט, מחברים או מחסרים כפי הצורך.

בתכנית זו פועלים על הרשומה אשר נמצאת במאגר. בדוגמא הקודמת, בתכנית חישוב מידות חום, פעלנו על הרשומה AVODA שמוגדרת בתכנית. אפשר גם לקרוא את הרשומה לשטח עבודה בתכנית, אך לשם כך יידרשו יותר משתנים ופעולות, כמו שנראה להלן:

```
(1)    PROGRAM MELAI2 (PEULOT, OUTPUT);
(2)    CONST
(3)          DUMMY = 'ZZZZZZ';
(4)    TYPE
(5)      CODE = PACKED ARRAY [1..6] OF CHAR;
(6)      PEULA = RECORD
(7)                 ZEHUT_PARIT : CODE;
(8)                 SUG_PEULA : (MECHIRA, KABALA, HACHZARA);
(9)                 CAMUT : INTEGER
(10)              END;
(11)   VAR
(12)     PEULOT : FILE OF PEULA;
(13)     SHINUI : INTEGER;
(14)     PARIT_NOCHCHI : CODE;
(15)     AVODA : PEULA;
(16)   BEGIN
(17)     RESET (PEULOT);
(18)     WRITELN ('מספר פריט ^^^^^ שינוי במשך החודש');
(19)     WRITELN;
(20)     READ (PEULOT, AVODA);
(21)     WITH AVODA   DO
(22)         WHILE (ZEHUT_PARIT <> DUMMY) DO
(23)           BEGIN
(24)             PARIT_NOCHCHI := ZEHUT_PARIT;
(25)             SHINUI := 0;
(26)             WHILE (ZEHUT_PARIT = PARIT_NOCHCHI) AND
(27)                 (ZEHUT_PARIT <> DUMMY) DO
```

```
(28)            BEGIN
(39)              CASE SUG_PEULA OF
(30)                MECHIRA : SHINUI:= SHINUI - CAMUT;
(31)                KABALA, HACHZARA : SHINUI:= SHINUI + CAMUT
(32)              END;
(33)              READ (PEULOT, AVODA)
(34)            END;                         (* פנימי WHILE..DO *)
(35)           WRITELN (SHINUI, '^':12, ZEHUT_PARIT:9)
(36)        END                              (* חיצוני WHILE..DO *)
(37)    END.
```

הסבר:

(20), (33) מבצעים פעמיים את הפרוצדורה READ. בפעם הראשונה קוראים את הרשומה הראשונה בקובץ לתוך הרשומה AVODA כדי לעבד את השדות שלה בלולאה (22)-(36). לאחר סיום הלולאה קוראים רשומה נוספת כדי שרשומה חדשה תהיה מוכנה לעיבוד כאשר נכנסים שוב פעם ללולאה.

בדוגמא זו מניחים שהרשומה האחרונה מכילה ערך DUMMY בשדה ZEHUT_PARIT כדי לציין שמגיעים לסוף הקובץ. בשיטה זו נוהגים בגירסאות של פסקל שבהן EOF איננו פועל בקבצים רגילים. אם הרשומה שקוראים בשורה (33) היא רשומת הדמה, לא נכנס שוב ללולאה ונסיים את התכנית.

11.3 פרוצדורות מינהלה של קבצים (*)

חוקי שפת פסקל דורשים שכאשר קובץ מועבר לפרוצדורה כפרמטר, הוא יהיה פרמטר משתנה. הסיבה לכך היא שאם הקובץ היה מועבר כמשתנה ערכי, היינו צריכים להעתיק את כל הקובץ לשטח העבודה של הפרוצדורה ולבזבז זמן רב וגם שטח החסנה. בנוסף, פקודת השמה איננה מוגדרת כלל עבור קבצים ולכן אסור לכתוב A=B אם A ו-B הם קבצים ולצפות שהתוכן של קובץ B יועבר לקובץ A. העברת נתונים לפרמטר ערכי כרוכה בהשמה של הנתונים לפרמטר הערכי וכאמור פקודת השמה אסורה בקבצים.

11.3.1 העתקת קבצים

הצהרות גלובליות

```
TYPE
          KOVETZ_PRITIM = FILE OF PARIT;
```

הפרוצדורה

```
PROCEDURE HAATEK_KOVETZ (VAR KOVETZ_YASHAN,
                             KOVETZ_CHADASH:
                             KOVETZ_PRITIM);
BEGIN
   RESET (KOVETZ_YASHAN);
```

```
(6)        REWRITE (KOVETZ_CHADASH);
(7)        WHILE NOT EOF (KOVETZ_YASHAN)  DO
(8)           BEGIN
(9)              KOVETZ_CHADASH^ := KOVETZ_YASHAN^;
(10)             PUT (KOVETZ_CHADASH);
(11)             GET (KOVETZ_YASHAN)
(12)          END
(13)    END;
```

הסבר:

א. אם KOVETZ_YASHAN הוא ריק, הפקודה RESET(KOVETZ_YASHAN) בשורה (5) תגרום שהתשובה ל-EOF תהיה TRUE. זה ימנע מן התכנית להכנס בזמן הביצוע אל הלולאה בשורה (7).

ב. המשתנה PARIT יכול להיות מכל סוג, למשל INTEGER, REAL או רשומה.

ג. אם משתמשים בפרוצדורות READ ו-WRITE אפשר להחליף את השורות (9)-(11) בשתי השורות הבאות:

```
READ (KOVETZ_YASHAN, AVODA);
WRITE (KOVETZ_CHADASH, AVODA);
```

הפקודה READ מעבירה נתונים מן הקובץ KOVETZ_YASHAN אל הרשומה AVODA, שיש להצהיר עליה כסוג איבר של הקובץ, מסוג PARIT. הפקודה WRITE מעבירה נתונים משטח AVODA ל-KOVETZ_CHADASH.

11.3.2 השוואה של שני קבצים

הצהרה גלובלית:

```
TYPE
   KOVETZ_PRITIM = FILE OF PARIT;
   TOTZAA = (SHAVIM, RISHON_KATAN, SHENI_KATAN, SHONIM)
```

הפונקציה:

```
FUNCTION HASHVAA (VAR KOVETZ1,KOVETZ2: KOVETZ_PRITIM): TOTZAA;
VAR
  MCHAPES : BOOLEAN;
BEGIN
  RESET (KOVETZ1);
  RESET (KOVETZ2);
  MCHAPES := TRUE;
  WHILE MCHAPES  DO
    IF EOF (KOVETZ1) THEN
      BEGIN
        MCHAPES := FALSE;
        IF EOF (KOVETZ2) THEN
          HASHVAA := SHAVIM
        ELSE
          HASHVAA := RISHON_KATAN
```

```
(16)        END
(17)      ELSE
(18)        IF EOF (KOVETZ2) THEN
(19)           BEGIN
(20)              MCHAPES := FALSE;
(21)              HASHVAA := SHENI_KATAN
(22)           END
(23)        ELSE
(24)          IF KOVETZ1^  <> KOVETZ2^  THEN
(25)            BEGIN
(26)              MCHAPES := FALSE;
(27)              HASHVAA := SHONIM
(28)            END
(29)          ELSE
(30)            BEGIN
(31)               GET (KOVETZ1);
(32)               GET (KOVETZ2)
(33)            END
(34)  END;
```

הסבר:

פרוצדורה זו נותנת תוצאה "SHAVIM" רק אם שני הקבצים זהים לגמרי. "זהות" פרושה מספר שווה של רשומות ותוכן זהה בכל זוג רשומות, אחת מן הקובץ הראשון ואחת מן הקובץ השני.

במהלך ההשוואה של הקבצים אפשר לקבל תוצאות שונות:

א. שני הקבצים שווים. תשובה זו מקבלים כאשר מגיעים לסוף הקובץ הראשון ולסוף הקובץ השני באותו זמן, בשורה (13).
ב. אורכו של הקובץ הראשון קטן מאורכו של השני, אבל עד סופו של הקובץ הראשון, כל הרשומות היו זהות בשני הקבצים, בשורה (15).
ג. אורכו של הקובץ הראשון גדול מאורכו של השני, אבל עד סוף הקובץ השני כל הרשומות היו זהות, בשורה (21).
ד. הקבצים אינם שווים, בשורה (27).

המשתנה PARIT בשורה (2) של ההצהרות יכול להיות מוגדר כסוג פשוט של נתונים (INTEGER, REAL, BOOLEAN, CHAR), או סוג מבני (כמו ARRAY, RECORD וכו'). משתנה זה אינו יכול להיות קובץ.

11.3.3 הוספה לסוף הקובץ

כאשר דרוש להוסיף רשומות לקובץ קיים יש צורך לפתור בעיה לוגית שנובעת ממבנה השפה. בעת שפותחים קובץ לכתיבה (ע"י פקודת REWRITE), המצביע פונה למקום הראשון שאפשר לכתוב בו. מכאן, שאי אפשר להוסיף רשומות מבלי למחוק את כל הרשומות אשר קיימות כבר. הפתרון היחיד הוא להעתיק את כל הקובץ הקיים לקובץ חדש, להעתיק מן הקובץ החדש חזרה אל הקובץ המקורי ולהוסיף

בסופו את הרשומות החדשות. פרוצדורה אשר מבצעת פעולה זו מוצגת להלן:

<u>הצהרה גלובלית:</u>

```
TYPE
  KOVETZ_PRITIM = FILE OF PARIT;
```

<u>הפרוצדורה:</u>

```
PROCEDURE HACHANA_LEHOSAFA (VAR KOVETZ : KOVETZ PRITIM);
  VAR
    KOVETZ_MEKOMI : KOVETZ_PRITIM;
  PROCEDURE HAATEK_KOVETZ (VAR KOVETZ_YASHAN,
               KOVETZ_CHADASH : KOVETZ_PRITIM) ; EXTERNAL;
    BEGIN
      HAATEK_KOVETZ (KOVETZ, KOVETZ_MEKOMI);
      HAATEK_KOVETZ (KOVETZ_MEKOMI, KOVETZ)
    END ;
```

<u>הסבר:</u>

פרוצדורה זו מכינה את הקובץ לכתיבה. אין צורך לבצע REWRITE על KOVETZ אחרי הפעלת הפרוצדורה הזאת, ואם זה יבוצע, כל התוכן הנוכחי של הקובץ ימחק. פרוצדורה זו פועלת בהנחה שבפרוצדורה HAATEK_KOVETZ (עיין בפרוצדורה 11.3.1) לא סוגרים את הקובץ, מכיון שהפקודה CLOSE אינה נחוצה בפסקל סטנדרטי. לו סגרנו את הקובץ, היינו מוכרחים לפתוח אותו שוב בפקודת REWRITE והיינו נקלעים שוב פעם לבעיה שפקודה זו מוחקת את תוכנו של הקובץ.

11.3.4 מיזוג של קבצים ממויינים

<u>הצהרה גלובלית:</u>

```
TYPE
  PARIT = RECORD
            SADEH_MAFTEACH : ... ;
            SADEH1:
              :
          END;
  KOVETZ_PRITIM = FILE OF PARIT;
```

<u>הפרוצדורה:</u>

```
PROCEDURE MIZUG_KVATZIM (VAR KOVETZ_RISHON,
                             KOVETZ_SHENI,
                             KOVETZ_MEMUZAG : KOVETZ_PRITIM);
  BEGIN
    RESET (KOVETZ_RISHON);
    RESET (KOVETZ_SHENI);
    REWRITE (KOVETZ_MEMUZAG);
    WHILE NOT (EOF (KOVETZ_RISHON) OR EOF (KOVETZ_SHENI)) DO
```

```
(9)          BEGIN
(10)          IF KOVETZ_RISHON^.SADEH_MAFTEACH <
(11)                KOVETZ_SHENI^.SADEH_MAFTEACH  THEN
(12)             BEGIN
(13)                KOVETZ_MEMUZAG^ := KOVETZ_RISHON^;
(14)                GET (KOVETZ_RISHON)
(15)             END
(16)          ELSE
(17)             BEGIN
(18)               KOVETZ_MEMUZAG^  := KOVETZ_SHENI^;
(19)               GET (KOVETZ_SHENI)
(20)             END;
(21)          PUT (KOVETZ_MEMUZAG)
(22)        END;
(23)      WHILE NOT EOF (KOVETZ_RISHON) DO
(24)        BEGIN
(25)          KOVETZ_MEMUZAG^ := KOVETZ_RISHON^;
(26)          PUT (KOVETZ_MEMUZAG);
(27)          GET (KOVETZ_RISHON)
(28)        END;
(29)      WHILE NOT EOF(KOVETZ_SHENI)  DO
(30)        BEGIN
(31)          KOVETZ_MEMUZAG^:= KOVETZ_SHENI^;
(32)          PUT (KOVETZ_MEMUZAG);
(33)          GET (KOVETZ_SHENI)
(34)        END
(35)    END;
```

<u>הסבר:</u>

התכנית בנויה מ-3 לולאות:

א. הלולאה הראשונה בשורות (8)-(22) פועלת כל עוד לא מגיעים לסוף של אחד הקבצים. בשורות (10)-(11) בודקים אם הערך של שדה המפתח בקובץ הראשון קטן מהערך של השדה המקביל בקובץ השני. אם התשובה חיובית, כותבים את הרשומה הזאת בקובץ הממוזג, בשורות (13) ו-(21). אם שדה המפתח בקובץ הראשון גדול או שווה לשדה המקביל בקובץ השני, מבצעים את השורות (17) עד (20) ולאחריהן את שורה (21).

אם במקום שורה (13) נכתוב: WRITE (KOVETZ_MEMUZAG, KOVETZ_RISHON^);
ואם במקום שורה (18) נכתוב: WRITE (KOVETZ_MEMUZAG, KOVETZ_SHENI^);
נוכל להשמיט את שורה (21).

ב. הלולאה השנייה מופעלת אם הקובץ הראשון יותר ארוך מן הקובץ השני. במצב זה צריכים להעתיק את הרשומות הנוספות כמו שהן לקובץ הממוזג. גם כאן אפשר להחליף שורות (25)-(26) בפקודה:
WRITE (KOVETZ_MEMUZAG, KOVETZ_RISHON^)

ג. הלולאה השלישית מופעלת אם הקובץ השני יותר ארוך מן הראשון. במצב זה צריך להעתיק את הרשומות הנוספות לקובץ הממוזג. גם כאן אפשר להחליף את השורות (31)-(32) בפקודה: WRITE (KOVETZ_MEMUZAG, KOVETZ_SHENI^).

גם כאשר משתמשים בפרוצדורה WRITE, לא יוצרים רשומה מיוחדת כדי להחזיק את הקלט בזמן הקריאה. במקום זה קוראים בעזרת פקודת GET ומשאירים את הרשומה הנקראת בתוך המאגר.

רק אחת מן הלולאות ב' ו-ג' מתבצעת בכל ריצה של הפרוצדורה, שהרי מגיעים לסופו של אחד מן הקבצים קודם ואז אין משמעות להפעלת הלולאה השנייה.

11.4 מיון קבצים (*)

נניח שרוצים למיין את הקובץ KOVETZ_A שמכיל רשומות עם המפתחות הבאים:

52	58	42	26	62	39	37	59

כדי לבצע את המיון נוקטים את הצעדים הבאים:

א. מחלקים את רשומות הקובץ KOVETZ_A לשני קבצים: KOVETZ_B ו-KOVETZ_C. מתחילים את התהליך מן הרשומה הראשונה, אשר המפתח שלה הוא 52. ממשיכים עד שמגיעים לרשומה הראשונה שאינה בסדר עולה, אשר המפתח שלה הוא 42. שתי הרשומות המסודרות 52 ו-58 הן <u>סדרה</u> (Run) ומעבירים אותן לקובץ KOVETZ_B.

את הסדרה השנייה (הרשומה 42 לבד) מעבירים ל-KOVETZ_C. את הסדרה השלישית (רשומות 26 ו-62) מעבירים שוב ל-KOVETZ_B. את הסדרה הרביעית, רשומה 39, מעבירים לקובץ KOVETZ_C ואת הסדרה החמישית הכוללת את הרשומות 37 ו-59 מעבירים ל-KOVETZ_B.

לאחר החלוקה הראשונה של רשומות הקובץ מקבלים את שני הקבצים שתכולתם מתוארת בתרשים:

52 58	26 62	37 59	KOVETZ_B

42	39	KOVETZ_C

ב. ממזגים את שני הקבצים שנוצרו אל קובץ KOVETZ_A. כל זוג של סדרות ממזגים לסדרה אחת. מיזוג זה מקטין את מספר הסדרות בגורם של שניים. בדוגמא זו, כאשר ממזגים שני קבצים לקובץ KOVETZ_A מקבלים:

42 52 58	26 39 62	37 59	KOVETZ_A

מכאן ואילך חוזרים על צעדים א' ו-ב'. כלומר, מפצלים את KOVETZ_A לשניים ומקבלים (צעד א'):

KOVETZ_B

42	52	58	37	59

KOVETZ_C

26	39	62

ממזגים את שני הקבצים (צעד ב') ומקבלים קובץ אחד יותר מסודר:

KOVETZ_A

26	39	42	52	58	62	37	59

מפצלים שוב את הקובץ (צעד א') ומקבלים:

KOVETZ_B

26	39	42	52	58	62

KOVETZ_C

37	59

ממזגים (צעד ב') ומקבלים קובץ שבו כל הרשומות בסדר עולה.

KOVETZ_A

26	37	39	42	52	58	59	62

כאן מסיימים את המיון, כאשר מספר הסדרות אחרי המיזוג שווה ל-1.

כמה פעמים צריך לבצע את התהליך של פיצול ומיזוג?
המספר המקסימלי של סדרות הוא N, כאשר יש בקובץ N רשומות בלתי מסודרות לחלוטין. "אי סדר" זה הוא המקרה שבו הרשומות נמצאות בסדר מיון הפוך לסדר הרצוי. בכל פעם שמבצעים צעדים א' ו-ב' מחלקים מספר זה לשניים וכך אפשר להגיע לסדרה אחת אחרי X פיצולים של הקובץ המקורי, כאשר $N/2^x=1$ ומכאן נובע ש: $X=\log_2 N$. לפיכך, המספר המקסימלי של מעברים כולל $\log_2 N$ פיצולים ואותו מספר של מיזוגים.

המיון שהוצג מבוסס על תהליך חוזר של פיצול ומיזוג רשומות עד שמקבלים סדרת רשומות אחת ממויינת. <u>מעבר</u> (Pass) כולל פיצול אחד ומיזוג אחד.

הפרוצדורה המבוססת על שיטת מיון זו ניתנת להלן:

<u>הצהרות גלובליות:</u>

```
TYPE
   PRINT = RECORD
            MAFTEACH: 0..MAXINT;
            NETUNIM: ...
          END;
   KOVETZ_PRITIM: FILE OF PARIT;
VAR
     KOVETZ_A : KOVETZ_PRITIM;
```

<u>הפרוצדורה:</u>

```
PROCEDURE MIYUN(VAR KOVETZ_A: KOVETZ_PRITIM);
  VAR
      KOVETZ_B, KOVETZ_C: KOVETZ_PRITIM;
      MISPAR_SIDROT: INTEGER;
      SOF_SIDRA: BOOLEAN;

  PROCEDURE TAVIR_RESHUMA (VAR KOVETZ_IN,
                         KOVETZ_OUT: KOVETZ_PRITIM;
                         VAR SOF: BOOLEAN); FORWARD;

  PROCEDURE CHALUKA;
                  (* חלוקה של הרשומות של KOVETZ_A
           לשני קבצים: KOVETZ_B ו-KOVETZ_C. סדרה אחת נכתבת
  ב: KOVETZ_B וסדרה אחרת נכתבת ב: KOVETZ_C לסירוגין. *)
  BEGIN
     RESET (KOVETZ_A);
     REWRITE (KOVETZ_B);
     REWRITE (KOVETZ_C);
     REPEAT
        REPEAT           (* העתק סדרה מ-KOVETZ_A ל-KOVETZ_B *)
           TAVIR_RESHUMA (KOVETZ_A, KOVETZ_B, SOF_SIDRA)
        UNTIL SOF_SIDRA;
        IF NOT EOF(KOVETZ_A) THEN
           REPEAT        (* העתק סדרה מ-KOVETZ_A ל-KOVETZ_C *)
              TAVIR_RESHUMA (KOVETZ_A, KOVETZ_C, SOF_SIDRA)
           UNTIL SOF_SIDRA
     UNTIL EOF(KOVETZ_A)
  END;

  PROCEDURE MIZUG;
      BEGIN
        REWRITE (KOVETZ_A);
        RESET (KOVETZ_B);
        RESET (KOVETZ_C);
        MISPAR_SIDROT:=0;
        WHILE NOT (EOF(KOVETZ_B) OR EOF(KOVETZ_C)) DO
```

```
          BEGIN
            REPEAT
             IF KOVETZ_B^ < KOVETZ_C^  THEN
              BEGIN
               TAVIR_RESHUMA (KOVETZ_B, KOVETZ_A, SOF_SIDRA);
               IF  SOF_SIDRA  THEN
                 REPEAT
                  TAVIR_RESHUMA (KOVETZ_C, KOVETZ_A, SOF_SIDRA)
                 UNTIL  SOF_SIDRA
              END
             ELSE
              BEGIN
               TAVIR_RESHUMA (KOVETZ_C, KOVETZ_A, SOF_SIDRA);
               IF  SOF_SIDRA  THEN
                 REPEAT
                  TAVIR_RESHUMA (KOVETZ_B, KOVETZ_A, SOF_SIDRA)
                 UNTIL  SOF_SIDRA
              END
            UNTIL  SOF_SIDRA;
            MISPAR_SIDROT:= MISPAR_SIDROT+1
          END;
          WHILE NOT EOF (KOVETZ_B) DO
                   (* העתק את הסדרות האחרונות מ-KOVETZ_B
                                                    אל KOVETZ_A. *)
            BEGIN
              REPEAT
               TAVIR_RESHUMA (KOVETZ_B, KOVETZ_A, SOF_SIDRA)
              UNTIL  SOF_SIDRA;
              MISPAR_SIDROT:= MISPAR_SIDROT+1
            END;
          WHILE NOT EOF(KOVETZ_C) DO      (* העתק את הסדרות
                              האחרונות מ-KOVETZ_C ל-KOVETZ_A. *)
            BEGIN
              REPEAT
               TAVIR_RESHUMA (KOVETZ_C, KOVETZ_A, SOF_SIDRA)
              UNTIL  SOF_SIDRA;
              MISPAR_SIDROT:= MISPAR_SIDROT+1
            END
        END;

   PROCEDURE TAVIR_RESHUMA;
      VAR
          AVODA: PARIT;
      BEGIN
         READ (KOVETZ_IN, AVODA):
         WRITE (KOVETZ_OUT, AVODA);
         IF EOF(KOVETZ_IN) OR (KOVETZ_IN^ < AVODA)  THEN
            SOF:= TRUE
         ELSE
            SOF:= FALSE
```

```
(83)        END;
                                        (* התכנית הראשית - מיון *)
(84)      BEGIN
(85)        REPEAT
(86)          CHALUKA;
(87)          MIZUG
(88)        UNTIL MISPAR_SIDROT = 1
(89)      END;
```

<u>הסבר:</u>

(85)-(88) הלולאה העיקרית של התכנית הראשית מורכבת משני צעדים:

א. חלוקת הרשומות של קובץ אחד KOVETZ_A לפי סדרות לשני קבצים: KOVETZ_B ו-KOVETZ_C.

ב. מיזוג הסדרות לפי זוגות, כאשר סדרה אחת בכל זוג היא מ-KOVETZ_B וחברתה מ-KOVETZ_C.

ממשיכים לבצע את הלולאה עד אשר מבצעים את המיזוג האחרון, שבו מספר הסדרות שווה ל-1 והתהליך פוסק.

(9)-(26) בפרוצדורה CHALUKA מעבירים רשומות מן הקובץ המקורי לאחד מקבצי העזר עד שנוצר מצב שבו SOF_SIDRA שווה ל-TRUE, שורה (20). הדבר קורה כאשר פוגשים ברשומה אשר שדה המיון שלה קטן משדה המיון שברשומה הקודמת, שורות (79)-(80). במצב זה מעבירים רשומות מן הקובץ המקורי לקובץ העזר האחר, שורה (23), עד שפוגשים ברשומה עם שדה מיון קטן יותר.

עתה עוברים שוב אל הקובץ הראשון וממשיכים את התהליך לסירוגין עד שלא נשארת אף רשומה בקובץ המקורי.

(27)-(72) בפרוצדורה MIZUG מעבירים סדרות של רשומות משני הקבצים אל הקובץ הקרוי KOVETZ_A, שורות (38) ו-(46).

בתחילה משווים את המפתח שברשומה הראשונה בסדרה הראשונה שב-KOVETZ_B עם המפתח שברשומה הראשונה בסדרה הראשונה ב-KOVETZ_C. את הרשומה עם המפתח הקטן מביניהם מעבירים ל-KOVETZ_A (שורות 38, 46). חוזרים על התהליך הזה עד שמגיעים לסוף סדרה ב-KOVETZ_B או ב-KOVETZ_C (39, 47). אם מגיעים לסוף סדרה, למשל ב-KOVETZ_B (לאחר שהרשומה ב-KOVETZ_B היתה קטנה מהרשומה שמשווים אליה מ-KOVETZ_C), פירושו שכל הרשומות שנשארו בסדרה המקבילה ב-KOVETZ_C גדולות מזו שעכשיו קראנו מ-KOVETZ_B. נוכל להעתיק את הרשומות האלו ל-KOVETZ_A והן תצטרפנה לסדרה החדשה (39-42). תוצאה דומה תתקבל אם מגיעים לסוף קובץ לאחר קריאת רשומה מ-KOVETZ_C (47-50).

אם מגיעים לסוף אחד מן הקבצים ויש עוד סדרות בקובץ השני, מעתיקים אותן ל-KOVETZ_A, שורות (55)-(63) ו-(64)-(71).

התהליך עומד לפני סיום כאשר גם KOVETZ_B וגם KOVETZ_C מורכבים כל אחד מסדרה אחת. במקרה כזה, אחד משני דברים קורה:

א. מגיעים לסוף KOVETZ_B בשורה (38) ואז מעבירים ל-KOVETZ_A את השארית של הסדרה שב-KOVETZ_C (שורה 41). מוסיפים 1 ל-MISPAR_SIDROT, שעד עכשיו היה 0 (שורה 53) ויוצאים מן הלולאה.

ב. מגיעים לסוף KOVETZ_C בשורה (46), מעבירים ל-KOVETZ_A את השארית של הסדרה שב-KOVETZ_B (49), מוסיפים 1 ל-MISPAR_SIDROT (53) ויוצאים מן הלולאה.

בשני המקרים הפרוצדורה MIZUG מסתיימת וגם הלולאה של שורות (85)-(88) מסתיימת כי MISPAR_SIDROT=1. בזה סיימנו את המיון.

11.5 תרגילים

11.5.1 שאלות

(1) בתכנית MELAI1 תניח שבנוסף לשדה ZEHUT_PARIT יש שדה שנקרא CHANUT. הקובץ ממויין לפי CHANUT בתוך ZEHUT_PARIT. בכל ערך של ZEHUT_PARIT הפעולות מסודרות בסדר עולה של השדה CHANUT. יש לשנות את התכנית כדי שתדפיס שינוי במלאי בכל פריט בכל חנות ובסוף הטיפול בכל פריט תדפיס את השינוי שחל בכמות שלו בכל החנויות יחד.
מוצע להגדיר משתנה שיחזיק את מספר החנות ומשתנה שיחזיק את מספר הפריט ולבדוק לשינויים בכל אחד מהם.

(2) כתוב תכנית, או פרוצדורה, שתתבסס על הפרוצדורה MIZUG_KVATZIM. תכנית זו תקבל שמות של 2≤N≤5 קבצים ממויינים, שנזין כקלט ותמזג את כולם לקובץ פלט אחד.

(3) שנה את הפרוצדורה MIYUN כדי שתקרא קבוצות של 100 רשומות מקובץ A, תמיין אותן במיון פנימי בזיכרון המחשב בשיטת החלפה ותכתוב אותן לקבצים B ו-C לפי סדרות כפי שנעשה בפרוצדורה CHALUKA. לאחר מכן יתחיל המיון בשיטה המתוארת בפרק זה. המיון המוקדם יאיץ את המיון העיקרי, כי הוא מבטיח סדרות התחלתיות של לפחות מאה רשומות.

11.5.2 תכניות

(1) בנה קובץ סטודנטים, שמכיל את הרשומות לדוגמא שרשומות בטבלה.

מספר זהות	שם מלא	מין	מספר שעות	ממוצע
110000	CHAYA LIBERMAN	N	100	88.7
220000	SHLOMO ARTZI	Z	70	91.9
330000	YITZCHAK SHALEV	Z	84	53.8
440000	CHAYIM CHEFETZ	Z	100	71.6
550000	RUTH HALEVY	N	13	68.6
660000	CHAVA ZALTZMAN	N	120	61.9
770000	RUTH COHEN	N	78	93.2
880000	RUBI PINCHUK	Z	52	77.9
990000	URI SHEMESH	Z	84	81.3

(2) כתוב תכנית שתסרוק את הקובץ שבנית בשאלה הקודמת. הדפס את השמות של הבנים שיש להם ממוצע למטה מ-75 ואת השמות של הבנות שיש להן ממוצע מעל ל-75.

(3) הרשומות בקובץ העובדים במפעל מכילות את הנתונים הבאים:

משכורת חודשית	מצב משפחתי	מין	תואר	מקצוע	שם	מספר

עליך לכתוב את התכניות והפרוצדורות הבאות:

א. תכנית לבניית הקובץ.
ב. פרוצדורה לשליפת השמות של כל העובדים הזכרים.
ג. פרוצדורה לשליפת השמות של כל העובדים שמשכורתם למעלה מ-900 שקל.
ד. פרוצדורה לשליפת השמות של כל הפועלים שמשכורתם גבוהה מן המשכורת הממוצעת במפעל.
ה. פרוצדורה לשליפת השמות של כל העובדים שהם:
- מתכנתים בעלי תואר ראשון, או
- עובדים שמספר הזיהוי שלהם גדול מ-1500 והם רווקים.

(4) כתוב תכנית שמעבדת קובץ של רשומות עבור קבוצות בליגה הארצית לכדורגל. כל רשומה מורכבת מהשדות הבאים:
- שם הקבוצה,
- מספר המשחקים ששיחקה עד עכשיו,
- מספר המשחקים בהם ניצחה (3 נקודות עבור ניצחון),
- מספר המשחקים בהם הפסידה (0 נקודות).
- מספר המשחקים שהסתיימו בתיקו (נקודה אחת לכל קבוצה).
- סכום הנקודות המצטבר.

התכנית קוראת את קובץ הקלט אל מערך ומקבלת כקלט נוסף את התוצאות של משחקי השבוע. היא מעדכנת את הנתונים שבמערך וממיינת אותו בסדר יורד, כאשר שדה המיון הוא סכום הנקודות המצטבר. בסיום, התכנית כותבת את המערך המעודכן לקובץ ומדפיסה אותו.

(5) חזור על התכנית בתרגיל מספר (3) בפרק הקודם. תניח עכשיו שהנתונים של הלקוחות ושל הפעולות מוחזקים בקבצים. כתוב תכנית כדי לעדכן את קובץ הלקוחות ולהדפיס את הדו"ח הדרוש.

(6) חברה מוכרת 4 סוגים של מיץ.
נתוני המחירים מוחזקים בקובץ בן 4 רשומות, כאשר יש רשומה אחת עבור כל סוג של מיץ. הנתונים רשומים בטבלה:

מ י ץ	מחיר לפי ארגזים			
	1	6	12	24
תפוזים	5	29	56	110
תפוחים	6	35	68	131
לימון	8	47	90	170
ענבים	9	52	102	195

דוגמא לחישוב המחיר: אדם שקנה 9 ארגזים של מיץ תפוזים ישלם 44 שהם 3*5+29.

הזמנה של לקוח כוללת את הנתונים הבאים:
- מספר ההזמנה.
- שם לקוח.
- כתובת.
- עיר ומיקוד.
- סוג המיץ וכמות הארגזים (אפשר להזמין מספר סוגי מיץ).

......

END 0 (לציון סוף ההזמנה של הלקוח).

כתוב תכנית כדי להכין חשבונות עבור הלקוחות.

(7) מכינים אינדקס לספר. הרשומות הן במבנה אשר מתואר בהמשך ומוחזקות בקובץ נפרד לכל פרק.

דף	מלה באנגלית

כדי להפיק את האינדקס יש לעשות את הפעולות הבאות:
א. למיין את הרשומות עבור כל פרק.
ב. למזג את הקבצים עבור כל הפרקים לקובץ אחד.
ג. למזג יחד את כל הרשומות של אותה מלה, כדי שבקובץ הסופי תהיה שורה מסכמת אחת לכל מלה. אפשר להניח מקסימום של 20 איזכורים למלה.
ד. להדפיס את האינדקס במבנה הבא: BOOK 1,17,84 וכו'.

לדוגמא, אם רשומות אינדקס של המלה BOOK נראות כך:

1	BOOK

17	BOOK

84	BOOK

הרשומה המסכמת למלה BOOK היא:

BOOK	1	17	84	...	...

פרק 12

עדכון קבצים

12.1 עיבוד קבצים

בפרק הקודם הצגנו מספר פעולות בסיסיות בקבצים כגון הדפסה, מיון, השוואה והוספת רשומות. בפרק זה נציג את פעולות העיבוד העיקריות שמבצעים בקבצים - את פעולות העדכון (Update) השונות:

- הוספת רשומה.
- ביטול רשומה.
- שינוי ערך של שדה ברשומה.

בדרך כלל רוצים לבצע כמה פעולות באותה רשומה, או ברשומות שונות. לכן נהוג לבנות קובץ תנועות (Transactions File), אשר מכיל את כל הפעולות שרוצים לבצע בקובץ. הקובץ המקורי שאותו מעדכנים נקרא קובץ אב (Master File), קובץ הפלט המעודכן נקרא קובץ בן, או קובץ אב חדש (New Master File). זהו נוהל עיבוד באצווה (Batch), לעומת נוהל עיבוד מקוון (On-Line) שבו מעבדים כל תנועה בנפרד, כאשר היא מגיעה למחשב.

נניח שנתונים שני קבצים:
א. קובץ אב MASKOROT, אשר מכיל את הנתונים אודות עובדים במפעל,
ב. קובץ תנועות עדכון, שבו יש רשומות לביצוע פעולות שונות.

פעולות העדכון בקובץ התנועות הן:

א. הוספה של רשומה חדשה לקובץ.
התו H מציין שזו תנועה להוספת רשומה. התו H לא יופיע ברשומה החדשה שתכתב בקובץ MASKOROT.

ציון	שכר	H	שם העובד

ב. ביטול רשומה מהקובץ.
ברשומה זו דרוש שם העובד בלבד כדי לזהות את הרשומה שמבקשים למחוק מן הקובץ. התו M מציין את סוג התנועה.

שם העובד	M

ג. תנועה לעדכון שדה ברשומה.
תנועה זו מטפלת בשני עדכונים שונים: כדי לעדכן את תוכן השדה SCHAR, יש לכתוב S כסוג הפעולה וכדי לעדכן את השדה TZIYUN יש לכתוב T כסוג פעולה.

שם העובד	S	שכר חדש

שם העובד	T	ציון חדש

12.2 רשומות באורך משתנה

עד עכשיו טפלנו בקבצים עם רשומות במבנה ובאורך קבוע. אפשר גם לבנות קובץ עם רשומות באורך משתנה (Variant Records), כדוגמת הרשומות בקובץ התנועות שתואר קודם. מספר וסוג השדות יכולים להיות שונים בכל סוג רשומה.

בתכנית הבאה קולטים נתונים מהמסוף ובונים קובץ תנועות אשר מכיל רשומות מן הסוגים שהוצגו קודם.

```
PROGRAM MASKORET18 (INPUT, OUTPUT, TRANFILE);
CONST ORECH= 25;
TYPE
    SHEM = PACKED ARRAY[1..ORECH] OF CHAR;
    TRANSACTION= RECORD
                       SHEMOVED: SHEM;
                       PEULA: CHAR;
                       CASE CHAR OF
                       'H': (TSCHAR: REAL;
                             TTZIYUN: CHAR);
                       'M': ();
                       'S': (SCHAR_CHADASH: REAL);
                       'T': (TZIYUN_CHADASH: CHAR)
                     END;
VAR
      TRAN_CHADASH: TRANSACTION;
      TRANFILE: FILE OF TRANSACTION;
```

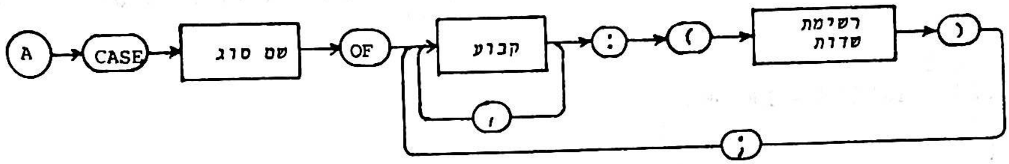
```
        ACHRON: SHEM;
        MAKOM: INTEGER;
BEGIN
  ACHRON := '^^^^^^^^^^^^^^^^^^^^^^^^^';     (* 25 תווי רווח *)
  REWRITE (TRANFILE);
  WHILE NOT EOF DO
    BEGIN
      WRITELN ('הקש שם וסוג פעולה');
      WITH TRAN_CHADASH DO
        BEGIN
          FOR MAKOM:= 1 TO ORECH DO
            READ (SHEMOVED[MAKOM]);
          READLN (PEULA);
          CASE PEULA OF
            'H':
               BEGIN
                 WRITELN ('הקש שכר, ציון^');
                 READLN (TSCHAR, TTZIYUN)
               END;
            'S':
               BEGIN
                 WRITELN ('הקש שכר');
                 READLN (TSCHAR, TTZIYUN)
               END
            'T':
               BEGIN
                 WRITELN  ('הקש ציון');
                 READLN (TZIYUN_CHADASH)
               END
          END;                          (* CASE *)
          IF SHEMOVED >= ACHRON  THEN
            BEGIN
              TRANFILE^:= TRAN_CHADASH;
              PUT (TRANFILE);
              ACHRON := SHEMOVED
            END
          ELSE WRITELN (SHEMOVED, 'לא במקום הנכון');
        END                        (* WITH *)
    END;                      (* WHILE..DO *)
    CLOSE (TRANFILE)
END.
```

(7) מגדירים את השדה PEULA. אילו הגדרנו בהצהרת TYPE (לפני הצהרת הרשומה) את סוגי הפעולות בדרך זו: SUGPEULOT=(H, M, S, T), אפשר היה לכתוב PEULA:SUGPEULOT ובהמשך לכתוב:

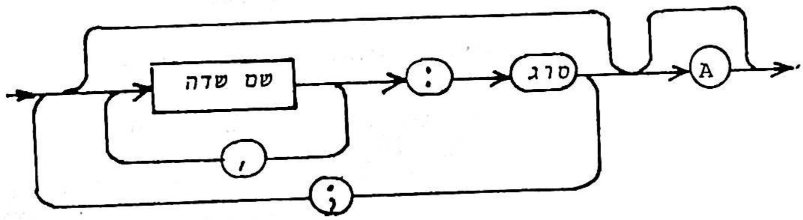
```
CASE SUGPEULOT OF
     (* בעבור H:... M:... S:... T... *)
END;
```

(8)-(13) מצהירים על החלק המשתנה של הרשומה. ברשומה משתנה תמיד קיים שדה משותף לכל רשומה אפשרית, שתוכנו קובע איך יראה המשך הרשומה שאחרי שדה זה, שדוגמא זו שמו: PEULA. רשומה משתנית כוללת חלק קבוע וחלק משתנה.

בהצהרה הבאה ניתן תאור של השדות שאפשר לכלול ברשומה, אחרי השדה המשותף:



הקבוע שווה לאחד מן הערכים של השדה המשותף PEULA. חוזרים עליו מספר פעמים כדי לטפל בכל הערכים האפשריים של השדה. רשימת השדות נראית כך:



החץ העליון ברשימת השדות מראה שהיא יכולה להיות ריקה, בדיוק כמו שדה 'M' בשורה (11). לא כל מהדר מתיר רשימה ריקה ועל כן האלטרנטיבה היא להשמיט כל התייחסות למקרה 'M'.

אסור לכתוב "CASE PEULA OF" במקום "CASE CHAR OF" (כי PEULA איננו שם סוג), אך מותר לשלב יחד שורות (7) ו-(8) ולכתוב "CASE PEULA:CHAR OF". הצהרה זו מקבילה להצהרה הקודמת: השדה PEULA יופיע בכל רשומה, כאשר השדה או שדות שבאים אחריו ישתנו כפונקציה של הערך שנמצא בשדה PEULA.

(31)-(47) בזמן מילוי השדות מבקשים נתונים אשר מתאימים לכל מקרה בנפרד.

(48)-(54) בודקים אם התנועות ממויינות. אם יש תנועה שמשבשת את הסדר, לא כותבים אותה בקובץ.
אפשר לייעל את התכנית ולכתוב קטע זה מיד לאחר קריאת השם, בשורה (29). אם הקלט לא יהיה בסדר המיון הנכון, אפשר יהיה לדלג על הקליטה של שאר הנתונים לפי(שורות (30)-(47)).יש לצפות לכך שהזנת הקלט תהיה לפי סדר המיון של התנועות ולכן התכנית נכתבה כפי שהיא מוצגת כאן.

12.2.1 שדות באורך משתנה (*)

רשימת השדות שהוצגה בציור של רשומה משתנית, יכולה גם כן להכיל רשומה

משתנית. הדבר מסביר את האופציה של חלק משתנה (A) בתוך התרשים של רשימת השדות. רשומה משתנית יכולה להופיע כשדה אחרון בלבד מתוך השדות הקשורים לערך קבוע מסוים של הצהרת CASE. ה-CASE החיצוני נסגר על ידי המלה END, שהיא בת הזוג של המלה RECORD. ה-CASE הפנימי נסגר על ידי הסוגר הימני שתוחם את רשימת השדות הקשורה לאותו ערך קבוע ששייך ל-CASE החיצוני.

לדוגמא נתאר רשומות של פציינטים אשר מקבלים טיפול במרפאת בית-החולים. הרשומה עבור פציינט ביתי, אשר בא לטיפול בלבד, שונה מזו של פציינט מאושפז. יש גם שוני בין רשומות של פציינטים מאושפזים: ישנם שמשלמים באופן פרטי וישנם שמשלמים באמצעות קופת-חולים.

```
TYPE
   TARICH = RECORD
              YOM: 1..31;
              CHODESH: 1..12;
              SHANA: 1985..2000
            END;
   SUG_CHESHBON = (PRATI, KUPA);
   SHEM = PACKED ARRAY[1..20] OF CHAR;
   RESHUMAT_PATIENT = RECORD
                        SHEM_PATIENT: SHEM;
                        GIL: 0..100;
                        CASE MEUSHPAZ: BOOLEAN OF
                          TRUE:
                            (MISPAR_CHEDER: INTEGER;
                            SHEM_ROFEH: SHEM;
                            DIETA: (BLI_MELACH, BLI_SUCAR, RAGIL);
                            CASE CHESHBON: SUG_CHESBON OF
                              PRATI: ( MAAMAD: (ASHIR, ANI));
                              KUPA: (MISPAR_ZEHUT: INTEGER;
                                     SHEM_HAKUPA: SHEM));
                          FALSE: (MISPAR_TELEPHONE: INTEGER;
                            TOR_HABA: TARICH)

                      END;
```

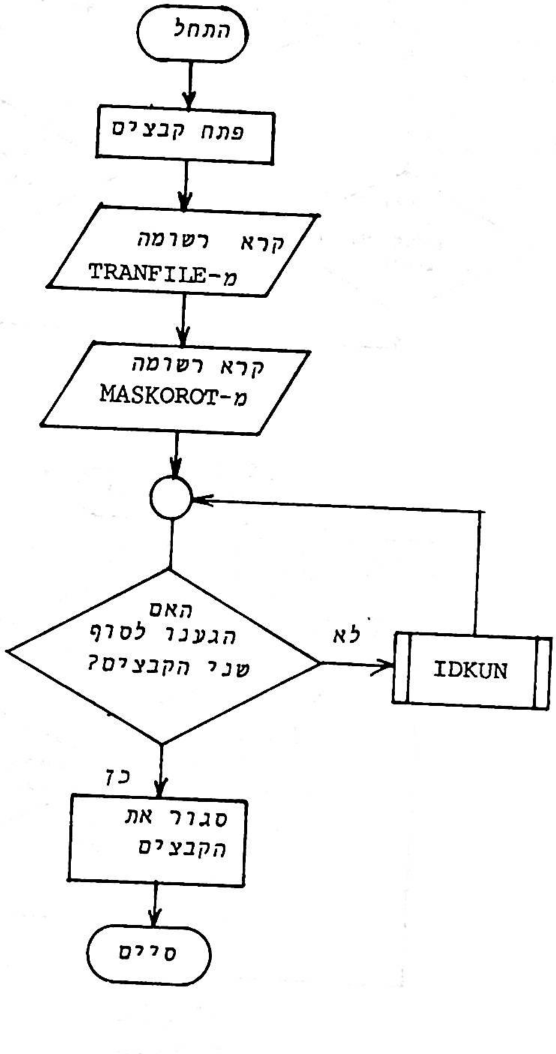
רשימת השדות הקשורה לכל קבוע מוקפת בסוגריים. לכן יש סוגר שמאלי פותח מיד אחרי TRUE, כאשר הסוגר הימני שלו מופיע מיד לפני FALSE ולפניו משמאל יש סוגר ימני נוסף ששייך ל-CASE הפנימי. בתוך הקבוע TRUE, בתחום ה-CASE הפנימי, יש סוגריים הקשורים עם הקבוע PRATI וסוגריים הקשורים עם הקבוע KUPA.

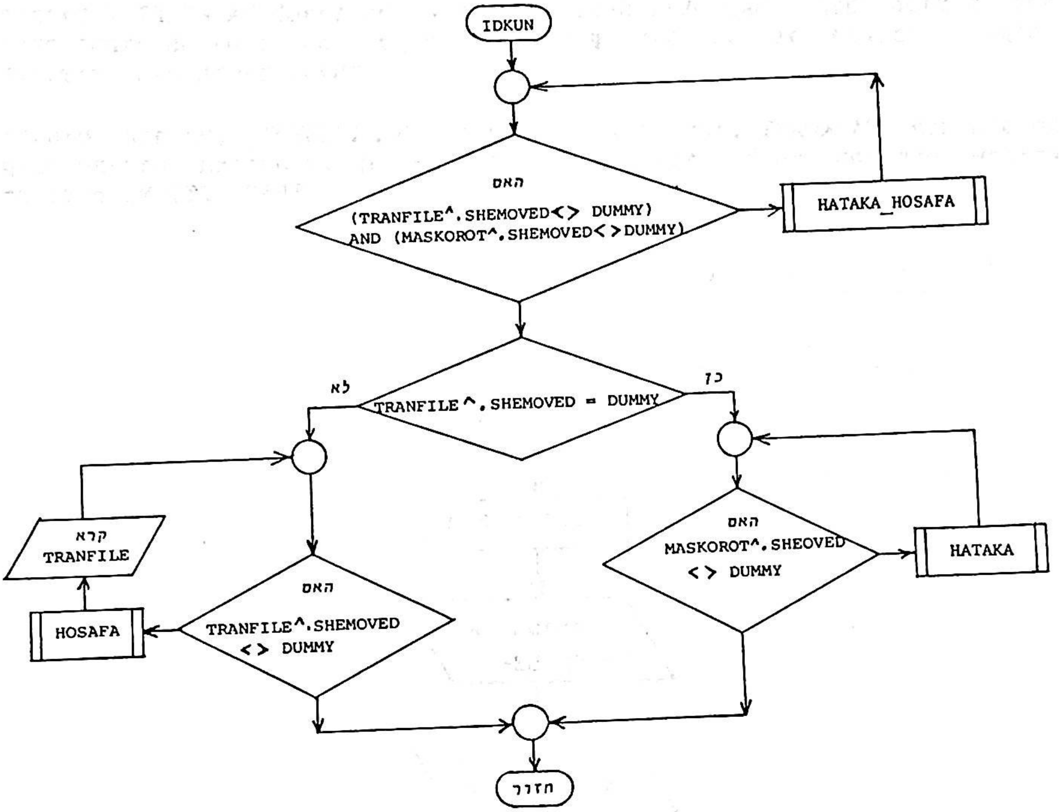
12.3 עדכון קובץ סדרתי

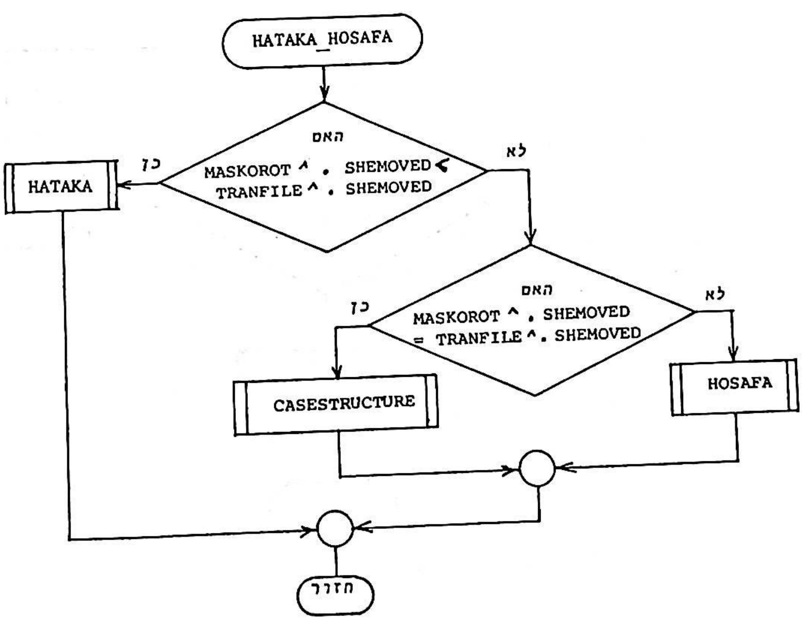
הקבצים שמטפלים בהם הינם <u>קבצים סדרתיים</u> (Sequential Files). כדי לעדכן או להוסיף רשומה בין הרשומות של קובץ סדרתי, או בסופו, צריך להעתיק אותו לקובץ שני. במהלך ההעתקה מעדכנים רשומות קיימות, מוסיפים רשומות, או מבטלים רשומות כנדרש. רשומות העדכון בקובץ התנועות חייבות להיות <u>ממויינות</u> לפי סדר המיון של הקובץ.

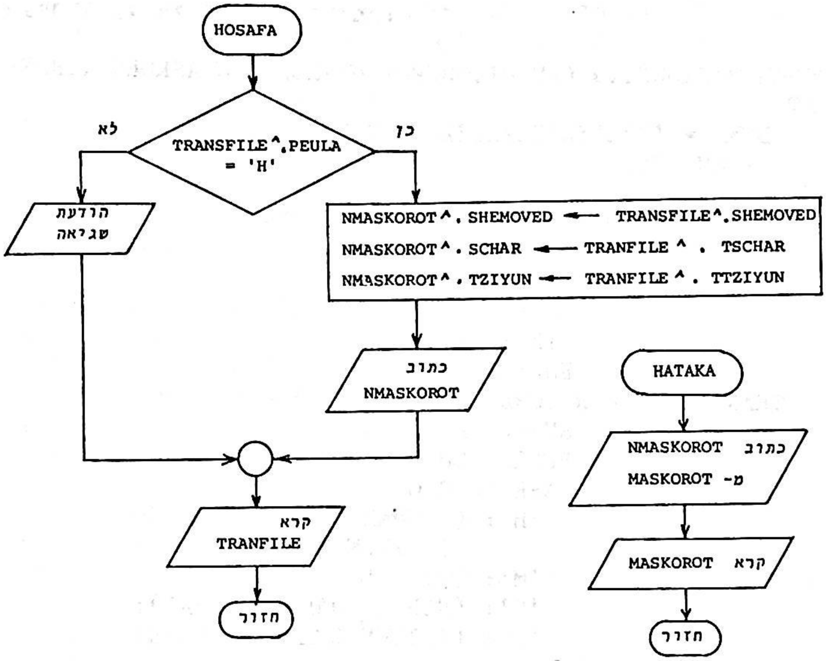
בתכנית MASKORET17 למדנו להקים קובץ אב וב-MASKORET18 - קובץ תנועות. עתה אפשר לכתוב תכנית שמטרתה לעדכן את קובץ האב על פי הנתונים שבקובץ התנועות. ראה תרשים זרימה.

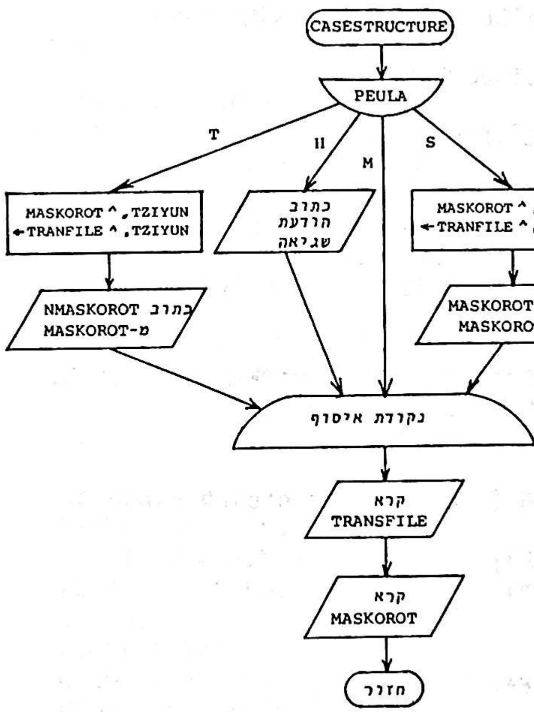
בתרשים הזרימה, MASKOROT הוא השם של קובץ-אב-ישן. NMASKOROT הוא השם של קובץ-אב-חדש, המבוסס על קובץ-אב-ישן, אשר מעדכנים על פי התנועות שבקובץ התנועות TRANSACTIONS.




IDKUN
האם
(TRANFILE^.SHEMOVED<> DUMMY)
AND (MASKOROT^.SHEMOVED<>DUMMY)
HATAKA_HOSAFA
לא
TRANFILE^.SHEMOVED = DUMMY
כן
קרא
TRANFILE
HOSAFA
האם
TRANFILE^.SHEMOVED
<> DUMMY
האם
MASKOROT^.SHEOVED
<> DUMMY
HATAKA
חזור


HATAKA_HOSAFA
האם
MASKOROT^. SHEMOVED<
TRANFILE^. SHEMOVED
כן
לא
HATAKA
האם
MASKOROT^. SHEMOVED
= TRANFILE^. SHEMOVED
כן
לא
CASESTRUCTURE
HOSAFA
חזור


HOSAFA
TRANSFILE^.PEULA = 'H'
לא
כן
הודעת שגיאה
NMASKOROT^.SHEMOVED ← TRANSFILE^.SHEMOVED
NMASKOROT^.SCHAR ← TRANFILE^.TSCHAR
NMASKOROT^.TZIYUN ← TRANFILE^.TTZIYUN
כתוב NMASKOROT
קרא TRANFILE
חזור
HATAKA
כתוב NMASKOROT
MASKOROT -מ
MASKOROT קרא
חזור


CASESTRUCTURE
PEULA
T
H
M
S
MASKOROT^.TZIYUN
←TRANFILE^.TZIYUN
כתוב
הודעת
שגיאה
MASKOROT^.SCHAR
←TRANFILE^.TSCHAR
NMASKOROT כתוב
MASKOROT-מ
MASKOROT כתוב
MASKOROT-מ
נקודת איסוף
קרא TRANSFILE
קרא MASKOROT
חזור

תכנית העדכון על פי תרשים הזרימה:

```
PROGRAM MASKORET19 (INPUT,OUTPUT,MASKOROT,NMASKOROT,TRANFILE);
CONST
      DUMMY= 'ZZZZZZZZZZZZZZZZZZZZZZZZZ';
      ORECH= 25;
TYPE
     SHEM= PACKED ARRAY[1..ORECH] OF CHAR;
     RESHUMATOVED= RECORD
                    SHEMOVED: SHEM;
                    SCHAR: REAL;
                    TZIYUN: CHAR
                   END;
     TRANSACTION= RECORD
                   SHEMOVED: SHEM;
                   PEULA: CHAR;
                   CASE CHAR OF
                     'H': (TSCHAR: REAL;
                           TTZIYUN: CHAR);
                     'M': ();
                     'S': (SCHAR_CHADASH: REAL);
                     'T': (TZIYUN_CHADASH: CHAR)
                  END;
VAR
     MASKOROT, NMASKOROT: FILE OF RESHUMATOVED;
     TRANFILE: FILE OF TRANSACTION;

PROCEDURE CASE_STRUCTURE;
BEGIN
   CASE TRANFILE^.PEULA OF
    'M': ;
    'S': BEGIN
           MASKOROT^.SCHAR:= TRANFILE^.TSCHAR;
           NMASKOROT^:= MASKOROT^;
           PUT (NMASKOROT)
         END;
    'T':BEGIN
          MASKOROT^.TZIYUN:= TRANFILE^.TTZIYUN;
          NMASKOROT^:= MASKOROT^;
          PUT (NMASKOROT)
        END;
    'H': WRITELN ('מנסים להוסיף רשומה קיימת')
   END;
   GET (TRANFILE);
   GET (MASKOROT)
END;

 PROCEDURE HOSAFA;
 BEGIN
   IF TRANFILE^.PEULA = 'H' THEN
      BEGIN
```

```
            NMASKOROT^.SHEMOVED := TRANFILE^.SHEMOVED;
            NMASKOROT^.SCHAR := TRANFILE^.TSCHAR;
            NMASKOROT^.TZIYUN := TRANFILE^.TTZIYUN;
            PUT (NMASKOROT)
          END
      ELSE
            WRITELN  ('פעולה בלתי אפשרית');
      GET (TRANFILE)
    END;

    PROCEDURE HATAKA;
    BEGIN
      NMASKOROT^ = MASKOROT^;
      PUT (NMASKOROT);
      GET (MASKOROT)
    END;

    PROCEDURE HATAKA_HOSAFA;
    BEGIN
      IF MASKOROT^.SHEMOVED < TRANFILE^.SHEMOVED THEN
         HATAKA
      ELSE
         IF MASKOROT^.SHEMOVED = TRANFILE^.SHEMOVED THEN
            CASE_STRUCTURE
         ELSE (* MASKOROT^.SHEMOVED > TRANFILE^.SHEMOVED *)
            HOSAFA
    END;

    PROCEDURE IDKUN;
    BEGIN
      WHILE (TRANFILE^.SHEMOVED <> DUMMY) AND
            (MASKOROT^.SHEMOVED <> DUMMY) DO
         HATAKA_HOSAFA;
      IF TRANFILE^.SHEMOVED = DUMMY THEN
         WHILE MASKOROT^.SHEMOVED <> DUMMY DO
            HATAKA
      ELSE (*MASKOROT^.SHEMOVED = DUMMY*)
         WHILE TRANFILE^.SHEMOVED <> DUMMY DO
            HOSAFA
    END;

    BEGIN                   (* תכנית ראשית *)
      REWRITE (NMASKOROT);
      RESET (TRANFILE);
      RESET (MASKOROT);
      IDKUN;
      CLOSE (TRANFILE);
      CLOSE (MASKOROT);
      CLOSE (NMASKOROT)
    END.
```

הסבר:

(93)-(85) התכנית הראשית.

(84)-(73) הפרוצדורה IDKUN: בפרוצדורה זו מבצעים את הלולאה העיקרית כל עוד שלא מגיעים לסוף של אחד מן הקבצים. מעבדים את הרשומה כאשר היא נמצאת במאגר ולא מעבירים אותה לרשומה שהוצהרה ע"י VAR. למשל, בשורה (75) פונים לשדה SHEMOVED ברשומת TRANFILE אשר נמצאת במאגר, על ידי הציון של המאגר ושם השדה: TRANFILE^.SHEMOVED.

כאשר מגיעים לסוף אחד מן הקבצים אפשר לזהות אחד משני מצבים ולפעול בהתאם:

א. סוף של קובץ התנועות:
מעתיקים את הרשומות שנשארו בקובץ-אב-ישן אל קובץ-אב-חדש.

ב. סוף של קובץ-אב-ישן:
מוסיפים את כל הרשומות שנשארו בקובץ התנועות כרשומות חדשות אל קובץ-אב-חדש, בתנאי שהן רשומות מסוג "H".

12.4 קבצים לגישה ישירה (*)

קבצים סדרתיים מאפשרים גישה סדרתית לרשומות. קובץ התנועות לעדכון של קובץ סדרתי חייב להיות ממוין לפי אותו סדר מיון. בעת העדכון מעתיקים את קובץ האב הישן וכותבים אותו מחדש עם העדכונים.

כאשר התנועות כבר ממויינות ומספרן גדול, אין לפעולת ההעתקה השפעה משמעותית על זמן העיבוד. הדבר שונה כאשר יש מספר קטן של עדכונים, או כאשר התנועות מגיעות באקראי. במקרה הראשון צריכים להעתיק קובץ שלם בעבור מספר קטן של שינויים ובמקרה השני צריכים לבצע עיבוד נוסף לפני העדכון: מיון של קובץ התנועות.

כאשר קובץ האב גדול ומספר תנועות העדכון קטן, כדאי יותר לעדכן רק את הרשומות הרלוונטיות ולא לקרוא ולכתוב את שאר הרשומות. דרישה זו קיימת במיוחד בעיבוד מקוון (On Line Processing), שבו יש מופע אקראי של תנועות בודדות לעדכון הקובץ או לקריאה של רשומות. בעיבוד זה צריך לתת שרות לתנועת העדכון באופן מיידי.

הגישה לרשומה בודדת אפשרית באמצעות שיטת הגישה הישירה (Direct Access Method). שיטת גישה זו אינה מוגדרת בשפת פסקל הסטנדרטית, אבל היא נוספה לה ע"י יצרני מחשבים ובתי תוכנה העוסקים בכתיבת מהדרים. הרחבה זו של השפה מאפשרת גישה ישירה לכל רשומה בקובץ לפי מספרה הסידורי, בלי שיהיה צורך לטפל ברשומות שקדמו לה בקובץ.

שיטת הגישה הישירה אפשרית רק באמצעי החסנה לגישה ישירה - DASD (Direct Access Storage Devices) כמו תקליטון או דיסק מגנטי, בעוד ששיטת הגישה הסדרתית אפשרית בכל אמצעי ההחסנה.

גישה לרשומה מסוימת מתאפשרת בעזרת <u>פקודת החיפוש SEEK</u>, אשר גורמת להזזה של ראש הקריאה/כתיבה של הדיסק אל הרשומה הדרושה:

SEEK (מספר רשומה, שם קובץ)

דוגמא לפתרון בעיה בשיטת הגישה הישירה:
נניח שאנשים רוצים לדעת את יתרת חשבונם בבנק, להפקיד או למשוך כסף מן החשבון. לשם כך צריך לטפל ברשומה של חשבון מסוים בלבד, מבלי לטפל בכל שאר רשומות החשבון. נניח שמבצעים עדכון של הסכום בלבד.

```
(1)   PROGRAM CHESHBON (CHESHBONOT, INPUT, OUTPUT):
(2)   VAR
(3)       CHESHBONOT: FILE OF REAL;
(4)       MISPAR_CHESHBON: INTEGER;
(5)       CONST: REAL;
(6)   BEGIN
(7)       RESET (CHESHBONOT);
(8)       WHILE NOT EOF  DO
(9)          BEGIN
(10)           WRITELN ('כתוב מספר חשבון');
(11)           READLN (MISPAR_CHESHBON);
(12)           WRITELN ('כתוב את הסכום עם סימן');
(13)           WRITELN ('להפקדה כתוב סימן + ולמשיכה כתוב סימן -');
(14)           READLN (SCHUM);
(15)           SEEK (CHESHBONOT, MISPAR_CHESHBON);
(16)           GET (CHESHBONOT);
(17)           CHESHBONOT^ := CHESHBONOT^ + SCHUM;
(18)           SEEK (CHESHBONOT, MISPAR_CHESHBON);
(19)           PUT (CHESHBONOT)
(20)         END
(21)  END.
```

<u>הסבר:</u>

(17)-(15) פקודת SEEK גורמת להזזה של ראש הקריאה של הדיסק לרשומה שמספרה שווה לערך של MISPAR_CHESHBON. כאשר מבצעים את הפרוצדורה GET בשורה (16), קוראים לתוך המאגר את הרשומה הזו. בשורה (17) מעדכנים את הסכום.

(18) לאחר הביצוע של ה-GET, ראש הקריאה עובר לרשומה MISPAR_CHESHBON+1. כאשר רוצים לכתוב את החשבון המעודכן במקום הנכון, צריך לעשות את ה-SEEK בפקודה (18). פקודה זו מחזירה את ראש הדיסק למקום שבו יש לכתוב את הרשומה שמספרה MISPAR_CHESHBON.

התכנית הזו כתובה עבור מהדר פסקל של UCSD. בכמה ניבים של שפת פסקל יש לפקודה SEEK פונקציות נוספות. בנוסף למיקום ראש הקריאה של הדיסק, היא קוראת את הרשומה הדרושה אל המאגר בלי לקדם את ראש הקריאה של הדיסק. במקרה זה יש למחוק מן התכנית את הפקודות GET ו-SEEK בשורות (16) ו-(18). מכיון שלא מבצעים את הפקודה GET בשורה (16) ולא מקדמים את ראש הקריאה כתוצאה מפקודה זו, אין גם צורך להחזיר אותו למקומו בעזרת פקודת SEEK בשורה (18).

בזמן כתיבת קובץ לגישה ישירה פותחים אותו בפקודת REWRITE, כנהוג בכל קובץ חדש. בזמן קריאה ועדכון של קובץ קיים, פותחים אותו בפקודת RESET ואז אפשר להשתמש בפקודת החיפוש SEEK כדי לאתר רשומות וכדי לקרוא ולכתוב בכל מקום בקובץ. פקודת RESET בדרך כלל מאפשרת רק קריאה (ולא כתיבה) וגם זה, רק באופן סדרתי החל מראש הקובץ.

אפשר לחפש רשומה (Search) לפי שדה המפתח (Key) שלה, אשר במקרה הנוח ביותר מתאים לכתובתה בדיסק, או למקומה היחסי בקובץ, החל מהכתובת של הרשומה הראשונה באותו קובץ. בהתאם לכך, חיפוש של רשומה יהיה פנייה אל כתובת בדיסק אשר שווה לשדה המפתח שלה. שדה המפתח אינו חייב להופיע ברשומה כי הוא ידוע על פי המיקום היחסי של כל איבר בקובץ, כמו בדוגמא הקודמת.

12.5 קובץ אינדקס סדרתי (*)

תחום המספרים של שדה המפתח אינו נמצא תמיד בתחום הכתובות שהוקצו לקובץ. תחום זה אינו מתחיל תמיד ב-1, אלא לדוגמא נמצא בתחום שבין 6000 לבין 1,000,000. יתכן גם שהמפתח אינו מספר אלא מחרוזת ואז הוא אינו יכול לציין את המיקום של הרשומה בקובץ. פותרים בעיה זו בעזרת טבלה של מספרים סידוריים מ-1 עד המספר של הרשומה האחרונה, כשליד כל אחד מהם רושמים את הערך של המפתח.

הטבלה מורכבת מרשומות במבנה הבא:

```
RESHUMAT_INDEX= RECORD
                  MISPAR_RESHUMA: INTEGER;
                  MAFTEACH: SUG_MAFTEACH
                END;
```

מן הרשומות בונים מערך (או טבלה), שסוגו מוגדר כך:

```
TAVLAT_INDEX= ARRAY [-1..N] OF RESHUMAT_INDEX
```

מבנה הטבלה:

	מפתח	מס' רשומה	מקום בטבלה
נתונים כלליים }		1	-1
}		6	0
	A17	5	1
	A84	2	2
	B14	7	3
	C17	4	4
	C35	6	5
	D40	1	6
רשומה שבוטלה }	ZZZ	3	7

בדוגמא זו מיספור הרשומות מתחיל ב-1. בפסקל של UCSD המיספור מתחיל ב-0.

את המשתנה INDEX מגדירים כך:

```
VAR
   INDEX: TAVLAT_INDEX
```

בטבלה שבדוגמא יש N+2 איברים. משתמשים בשניים הראשונים כדי להחזיק נתונים כלליים וב-N המקומות החל מן המקום השלישי כותבים מפתחות עם מספרי רשומה. באיבר INDEX[0].MISPAR_RESHUMA שבטבלה מחזיקים את מספר הרשומות הפעילות בקובץ. יתכן שיש יותר רשומות בקובץ אך הן מסומנות כמבוטלות ואין פונים אליהן יותר בתהליכי העיבוד.

כאשר מבטלים רשומות לא מוחקים אותן מהקובץ, אלא מוסיפים להן סימון של ביטול, כדי לציין שהן אינן פעילות יותר. באיבר INDEX[-1].MISPAR_RESHUMA רושמים את מספר הרשומות האלו, אשר עומדות לשימוש חוזר אם יהיה צורך בכך. מחיקה בפועל של רשומות אלו מתוך הקובץ תיעשה רק בזמן ארגון מחדש של טבלת האינדקס והעתקה של הקובץ למקום אחר באותו דיסק, או בדיסק אחר.

בטבלה שבדוגמא מספר הרשומות בקובץ הוא 7, אבל רק 6 מהן פעילות ולכן INDEX[0].MISPAR_RESHUMA=6. הרשומה השביעית, שמספרה הסידורי בקובץ הוא 3, נמצאת בסוף הטבלה עם מפתח ZZZ ולכן המקום של רשומה מספר 3 פנוי לשימוש חוזר. כאשר מוסיפים רשומות לקובץ אפשר יהיה לכתוב במקום זה רשומה חדשה ולעדכן את טבלת האינדקס בהתאם ולכן INDEX[-1].MISPAR_RESHUMA=1.

הקובץ שתואר בדוגמא הוא <u>קובץ אינדקס סדרתי</u> (Indexed Sequential File), אשר מאפשר עיבוד סדרתי וגם עיבוד בגישה ישירה. הפעולות שאפשר לעשות בקובץ זה הן:

- בניית הקובץ.
- עדכון רשומות.
- ביטול רשומות.
- הוספת רשומות.

א. בניית הקובץ.

כותבים את הקובץ בדיסק ורושמים בטבלה את המספרים הסידוריים של הרשומות והמפתחות שלהן. כאשר גומרים את הבנייה, ממיינים את הטבלה לפי המפתחות.

ב. עדכון רשומות.

משתמשים באינדקס כדי למצוא את המספר הסידורי של הרשומה שרוצים לעדכן. בעזרת הפקודה SEEK מאתרים אותה בקובץ ומוסיפים פקודה GET כדי לקרוא את הרשומה (אם כותבים בפסקל של UCSD). מעדכנים וכותבים אותה למקומה בקובץ. במקרה כזה גם מספר הרשומה וגם ערך המפתח לא משתנים. אם השינוי הוא של המפתח, צריך לעדכן גם את טבלת האינדקס ולמיין אותה מחדש.

ג. ביטול רשומות.

כדי לבטל רשומה יש לעשות את הצעדים הבאים:

1) לחפש את המפתח באינדקס.
2) לשנות את המפתח של הרשומה ל-'ZZZ'. לא משנים את מספר הרשומה כי הוא מציין את המקום הסידורי בקובץ אשר מיועד לשימוש חוזר.
3) למיין את טבלת האינדקס מחדש. גם הרשומה המבוטלת (שאינה פעילה) משתתפת במיון, מכיון שעוד לא שינינו את INDEX[0].MISPAR_RESHUMA. הרשומות שכבר מבוטלות נמצאות בסוף הטבלה, מכיון שהמפתח שלהן הוא 'ZZZ', ולכן הן אינן משתתפות במיון הנוכחי.

4) להקטין ב-1 את INDEX[0].MISPAR_RESHUMA מכיון שרשומה אחת הועברה ממצב פעילות למצב של אי-פעילות.
5) להוסיף 1 אל INDEX[-1].MISPAR_RESHUMA כי יש מקום פנוי נוסף של רשומה בקובץ שאפשר להשתמש בו לכתיבת רשומה חדשה.

בכל שלב, מספר הרשומות אשר בקובץ שווה לסכום:

INDEX[-1].MISPAR_RESHUMA + INDEX[0].MISPAR_RESHUMA

זהו המספר הכולל של הרשומות הפעילות והרשומות שהיו פעילות ובוטלו. אין לכך קשר לגודל טבלת האינדקס שהיא בת N+2 איברים, כי היא מיועדת לכל הרשומות שניתן לכתוב בקובץ.

בדוגמא זו, אם N הוגדר כקבוע ששווה ל-100, גודל הטבלה הוא 102. גודל הקובץ בפועל שווה למספר הכניסות בטבלה שהשתמשנו בהן, חוץ משני המקומות הראשונים שמיועדים ל"נתונים כלליים".

ד. הוספת רשומות.

כדי להוסיף רשומות יש לעשות את הצעדים הבאים:

1) לבדוק את INDEX[-1].MISPAR_RESHUMA כדי לדעת אם יש מקומות ריקים בקובץ.
2) אם יש מקומות ריקים, יש לחפש את הכניסה הראשונה שיש בה מפתח 'ZZZ'. הכתובת של כניסה זו בתוך האינדקס היא:

INDEX[0].MISPAR_RESHUMA + 1

בדוגמא זו נקבל 6+1=7 ולכן הכניסה המתאימה בטבלה היא INDEX[7]. כותבים את הרשומה החדשה במספר הרשומה שמופיע ב-INDEX[7] ובדוגמא - במקום הסידורי 3. מאתרים את המקום בדיסק בעזרת הפקודה SEEK וכותבים בעזרת הפקודה PUT. מעדכנים את המונים בשני המקומות הראשונים בטבלה ע"י הפקודות הבאות.

INDEX[-1].MISPAR_RESHUMA:= INDEX[-1].MISPAR_RESHUMA - 1
INDEX[0].MISPAR_RESHUMA:= INDEX[0].MISPAR_RESHUMA + 1

3) אם אין מקומות שהתפנו בקובץ כתוצאה של ביטול רשומות, כותבים את הרשומה החדשה בסוף הקובץ. בפסקל של UCSD מוצאים תחילה את הרשומה האחרונה בקובץ ואח"כ ממשיכים. יש לזכור, שבעת תכנות בפסקל UCSD, מיספור הרשומות בקובץ מתחיל ב-0.

הפקודה לחיפוש הרשומה היא: SEEK (INDEX[0].MISPAR_RESHUMA -1))
אח"כ מבצעים GET כדי לקרוא את הרשומה הזו וממשיכים בפקודה PUT כדי לכתוב את הרשומה החדשה.

בניבים אחרים של פסקל מבצעים SEEK ישר לסוף הקובץ (כאשר מיספור הרשומות מתחיל ב-1): SEEK (INDEX[0].MISPAR_RESHUMA +1))
אח"כ מבצעים PUT, עם הנתונים שנרשמו במאגר.
בשני המקרים מוסיפים 1 לאינדקס INDEX[0].MISPAR_RESHUMA וממיינים את האיברים בטבלה INDEX, חוץ משני האיברים הראשונים. מספר האיברים שממיינים שווה ל: INDEX[0].MISPAR_RESHUMA
במקרה זה: INDEX[-1].MISPAR_RESHUMA = 0
המיונים שהוזכרו מיועדים להכניס רק איבר אחד למקומו הנכון בתוך רשימה

ממויינת, ולכן נקטנו בשיטה של הזזת איברים במקום למיין את כל הרשומות.

מן הדיון הקודם אפשר להסיק שהאינדקס נושא נתונים חיוניים עבור השימוש בקובץ. לכן מאחסנים גם אותו בקובץ סדרתי בצורה זו: VAR

KOVETZ_INDEX: FILE OF TAVLAT_INDEX

בשפות עיליות המתמחות בטיפול בקבצים (למשל, COBOL) ובניבים מפותחים של פסקל, המהדר מטפל בעצמו בבניית טבלאות האינדקס הדרושות. המתכנת מציין רק את סוג הקובץ INDEX-SEQUENTIAL מבלי לטפל בטבלאות עצמן. בזמן קלט ופלט המתכנת מעביר את ערך המפתח לשדה מוגדר ברשומה ונותן את הוראת הקריאה, או הכתיבה בלבד.

12.6 תרגילים

12.6.1 שאלות

(1) הצהר על רשומה שמכילה את הנתונים הבאים:
- שם משפחה ושם פרטי.
- גיל.
- מצב משפחתי (רווק, נשוי, גרוש/אלמן).
- שם בן הזוג וגילו, אם מצב משפחתי הוא "נשוי".
- מספר הילדים, אם היה נשוי אי פעם.
- שמות וגיל הילדים, אם יש לו ילדים.

(2) קרא את השאלה (1) בפרק 10. הרחב את הצהרת הרשומה כדי לכלול נתונים נוספים. כלומר, הוסף שדות אם מתקיימים תנאים מסוימים של מצב משפחתי.
- אם רווק/ה - הוסף שדה בוליאני כדי לציין אם יש לו דירה.
- אם נשוי/אה - הוסף שדה לתאריך נישואין ושדה לשם בן הזוג.
- אם גרוש/ה - הוסף שדה לתאריך גירושין.
- אם אלמן/נה - הוסף שדה לתאריך הפטירה של בן הזוג.
- אם נישואין שניים - הוסף שדה לתאריך נישואין ראשונים.
- אם נישואין שניים - הוסף שדה לתאריך סיום נישואין ראשונים.
- אם נישואין שניים - הוסף שדה לתאריך נישואין שניים.
- אם נישואין שניים - הוסף שדה לשם בן הזוג הראשון.

12.6.2 תכניות

(1) השתמש בקובץ שבנית בתרגיל התכנות הראשון של פרק 11. עליך לעדכן אותו בהתאם לשעות הלימוד והציונים שקבלו הסטודנטים בשנה זו. בכל תנועת עדכון יופיעו הנתונים הבאים:
- מספר זהוי,
- שם סטודנט,
- מספר שעות של קורס שלמד,
- ציון שקבל בקורס.

השדות של מספר השעות בקורס והציון חוזרים כמספר הקורסים שהסטודנט למד במשך השנה. סדרת הזוגות "שעות" ו"ציון" מסתיימת כאשר יש 0 (אפס) בשדה מספר שעות.

כתוב תכנית שתעשה את הפעולות הבאות:

א. בדיקה אם יש טעות בנתונים: מספר זהות שאיננו נמצא בקובץ או שם שאיננו תואם את מספר הזהות שניתן.

ב. חישוב של מספר השעות החדש לכל סטודנט וממוצע הציונים החדש. למשל, אם לפני העדכון היו לסטודנט 75 שעות עם ממוצע של 80 ועכשיו למד 10 שעות וקיבל ציון של 100, הממוצע יהיה: (75*80+10*100)/85=82.3.

הקלט לתכנית:

מספר זהות	שם	שעות	ציון	שעות	ציון	שעות	ציון
100000	CHAYA LIBERMAN	3	81	5	88	2	65
200000	RUTH HALEVY	2	61	2	91	3	43
300000	BARUCH COHEN	5	93	2	100	3	88
400000	CHAVA ZALTZMAN	2	81	4	62	2	93
500000	URI HECKEL	4	75	2	85	3	80

(2) עליך לעדכן את הקובץ שבתרגיל 1. פעולות העדכון כוללות:
- הוספה של סטודנט חדש (H),
- מחיקה של סטודנט קיים (M),
- עדכון של מספר השעות ושל הממוצע (T).

ברשומות התנועות נמצאים השדות הבאים:
- מספר זהוי
- קוד (T,M,H)
- שאר הנתונים, אם דרוש.

כתוב תכנית שתבצע את העדכון הדרוש. השתמש בנתונים הבאים של קובץ התנועות.

שדה 1	שדה 2	שדה 3	שדה 4	שדה 5	שדה 6
000750	H	RUTH ZAMIR	N	25	89.5
200000	M				
288100	H	SHLOMO CARMELI	Z	78	65.4
555555	T	84	68.3		
900000	T	110	68.1		

(3) קובץ אב בבנק מורכב מרשומות המכילות מספרי חשבון של לקוחות ויתרות. קובץ התנועות מורכב מרשומות מספרי חשבון וסכום הפקדה (מספר חיובי) או סכום משיכה (מספר שלילי). יתכנו כמה פעולות עבור אותו מספר חשבון. עדכן את קובץ האב והדפס דוח שגויים עבור תנועות עם מספרי חשבון שאינם בקובץ האב.

(4) הרשומות בקובץ מלאי מכילות את השדות הבאים:
- מספר פריט.
- מלאי קיים.
- נקודת הזמנה.
- רמת מלאי רצויה.
- כמות בהזמנה.

מזמינים השלמת מלאי של פריט כאשר הסכום של המלאי הקיים ושל הכמות בהזמנה קטן מנקודת ההזמנה. הכמות המוזמנת היא ההפרש שבין הסכום של המלאי ושל ההזמנה הקיימת לבין רמת המלאי הרצויה.

מבנה רשומה בקובץ התנועות:
- מספר פריט.
- סוג פעולה (הוספה למלאי, הוצאה ממלאי).
- כמות.

בהנחה ששני הקבצים ממויינים לפי מספר הפריט, כתוב תכנית כדי לבצע עדכון שיתייחס לאפשרויות הבאות:
א. אין פעולות עבור פריט מסוים.
ב. יש פעולות אחדות עבור פריט מסוים.
ג. יש פעולות עבור פריט שאיננו קיים (ז"א יש טעות במספר פריט או בקובץ האב).

(5) עיין בתכנית מספר 2 בתרגילים של פרק 10. תוכל להניח שרשומות הסטודנטים מוחזקות בקובץ (ולא במערך) ובעבור כל קורס יש קובץ נפרד שמכיל את שמות הסטודנטים והציונים שלהם. קובץ הסטודנטים בכל קורס ממויין לפי שמות הסטודנטים.

עדכן את קובץ הסטודנטים על פי הנתונים שבקבצי הקורסים.

(6) עיין בתכנית מספר 6 בתרגילים של פרק 11.
בנוסף על קובץ המחירים יש קובץ לקוחות המכיל רשומות עם השדות האלה:
- שם הלקוח.
- מספר הלקוח.
- כתובת.
- עיר.
- סכום החוב.

התנועות מוחזקות בקובץ שלישי במבנה הבא:
- מספר הזמנה.
- מספר לקוח.
- סכום החיוב אם זו חשבונית, או סכום התשלום.
- קוד תנועה: חשבונית - C, תשלום - T.
- סוגים וכמויות של כל פריט שהזמין, אם זו חשבונית

כתוב תכנית כדי לעדכן את קובץ הלקוחות ולהדפיס חשבונית או קבלה.

(7) השתמש בקובץ המחירים שמתואר בתכנית מספר 6 בתרגילים של פרק 11, כדי לכתוב תכנית לעדכון. התכנית תטפל בשלושה סוגי עדכון:
א. עליה כללית של X% במחירים.
ב. קטגוריה חדשה של מחיר מוזל. המחיר החדש יכול להנתן בערכים מוחלטים, או ביחס מתמטי. יש למשל אפשרות לקנות 48 ארגזים ב-1.9 כפול המחיר של 24 ארגזים.
ג. הוספה, השמטה או שינוי ברשומה מסוימת של הקובץ.

(8) (*) בקובץ שמחזיק נתונים על הלקוחות של בנק מסוים, מבנה הרשומות הוא:

- מספר חשבון (10 מקומות).
- שם הלקוח (10 מקומות).
- יתרה.
- משיכת יתר מקסימלית מותרת (אפשר גם 0).

הלקוחות מבצעים הפקדות ומשיכות בעזרת מכשיר הבנקומט.

א. כתוב תכנית המשתמשת בגישה ישירה לקובץ הלקוחות כדי לעדכן את הרשומה של לקוח המבצע פעולות ולהדפיס אחת מן ההודעות הבאות:
- ההפקדה של (סכום) ע"י לקוח מספר (מספר) התקבלה.
- המשיכה של (סכום) ע"י לקוח מספר (מספר) מאושרת.
- המשיכה של (סכום) ע"י לקוח מספר (מספר) לא מאושרת, כי עבר על משיכת היתר המותרת לו.

ב. כתוב תכנית שמשתמשת בשיטת הגישה הישירה כדי לעדכן את קובץ הלקוחות. פעולות העדכון האפשריות הן:
- הוספה של לקוח חדש.
- ביטול רשומה של לקוח מהקובץ.
- עדכון שם או כתובת.

(9) (*) יש להרחיב את תכנית (4) לטיפול במלאי. רוצים לבנות מערכת מידע ניהולי שתאפשר להנהלה לדעת בכל זמן את מצב המלאי. כדי לעשות זאת יש לאפשר את הפעולות הבאות בגישה ישירה:

א. עדכון מיידי של הקובץ כאשר מתקבלת הזמנה של פריט מסוים או כאשר מנפקים פריט זה מהמלאי.
ב. הצגת מצב המלאי של פריט מסוים בתשובה לשאילתה.
ג. הצעה להנהלה להפסיק להחזיק בפריטים מסוימים אם הביקוש נמוך מ-10 בשלושה חודשים רצופים (כדי להגיע להחלטה כזו, צריך להוסיף תאריך בקובץ התנועות ולהעביר אותו לקובץ האב בעת עדכון המלאי).

פרק 13
קבצי תמליל

13.1 עיבוד של קובץ תמליל

קובץ תמליל (Text) הוא קובץ שאיבריו הם תווים ז"א, שכל רשומה בו היא בת תו אחד בלבד. לקובץ מסוג זה יש התכונות המקובלות של קבצים, אך יש לו תכונה נוספת חשובה: הרשומות, או התווים, מוחזקים בקבוצות, אשר המפריד ביניהן הוא תו מיוחד שנקרא גבול שורה (Line Boundary או Separator).

גבול שורה מאפשר חלוקה של קובץ תמליל לא רק לרשומות, כנהוג בכל קובץ, אלא גם לגושים של רשומות, אשר כל אחד מהם מקביל לשורה. השורות אינן חייבות להיות שוות באורכן. אורך השורה נקבע ע"י מיקומו של תו גבול השורה, אשר נמצא בשליטת המתכנת בעת כתיבה של התמליל, ע"י הפעלה של הפרוצדורה:

WRITELN(שם קובץ)

התכנית להלן מדגימה עיבוד של קובץ תמליל. היא סופרת את מספר הרווחים והתווים בקובץ תמליל נתון.

```
PROGRAM COUNT (TEXTFILE, OUTPUT);
CONST
      BLANK = '^';                        (* תו רווח *)
VAR
      TEXTFILE: TEXT;
      CH: CHAR;
      MISREV, MISTAV: INTEGER;           (* מונים למספר הרווחים *)
BEGIN                                    (* ולמספר התווים, בהתאמה *)
  MISREV:= 0;
  MISTAV:= 0;
  RESET (TEXTFILE);
  WHILE NOT EOF (TEXTFILE) DO
   BEGIN
     WHILE (NOT EOLN(TEXTFILE)) AND (NOT EOF(TEXTFILE)) DO
       BEGIN
         READ  (TEXTFILE, CH);
         IF CH = BLANK  THEN
             MISREV:= MISREV+1
```

```
(19)            ELSE  MISTAV:= MISTAV+1
(20)          END;
(21)        IF NOT EOF (TEXTFILE) THEN
(22)          READLN (TEXTFILE)
(23)         (* דלג על סימן סוף שורה והעבר את EOLN למצב FALSE *)
(24)     END;
(25)    WRITELN (MISREV, 'מספר הרווחים שווה^-^');
(26)    WRITELN (MISTAV, 'מספר התווים שווה ^-^');
(27)    CLOSE (TEXTFILE)
(28)  END.
```

<u>הסבר:</u>

(5) סוג המשתנה TEXTFILE הוא TEXT. TEXT הוא סוג סטנדרטי שמוצהר באופן אוטומטי כאילו המתכנת כתב: TEXT = FILE OF CHAR.

(11) כמו בקבצים אחרים, יש לפתוח ולסגור גם קובץ TEXT. יש מהדרים אשר סוגרים את הקובץ בסוף התכנית, באחרים - צריך המתכנת לעשות זאת (ראה שורה (27)).

(12) כדי לבדוק את התוכן של הקובץ, דרוש שהקטע הבא ימשיך להתבצע עד שמזהים סימן של סוף קובץ. כאשר סימן זה מועבר למאגר של הקובץ בזיכרון, הפונקציה הבוליאנית EOF(TEXTFILE) תקבל ערך TRUE (ראה גם שורה (21)).

(14)-(20) קטע זה ימשיך להתבצע בלולאה עד אשר מגיעים לסוף שורה, או עד לסוף הקובץ, לפי האירוע המוקדם יותר. כדי לגלות אם מגיעים לסוף שורה, משתמשים בפונקציה הבוליאנית: EOLN(שם קובץ). הערך של פונקציה זו הוא בדרך כלל FALSE חוץ מהמקרה שהמאגר מכיל סימן גבול שורה ואז הערך שלה הופך להיות TRUE.

(16) פרוצדורה זו קוראת איבר אחד (שגודלו תו אחד) מתוך TEXTFILE אל המשתנה CH. כאשר קוראים את התו שלפני גבול השורה, גבול השורה יגיע לחלון ויגרום לכך שהביטוי EOLN(TEXTFILE) יחזיר TRUE. כאשר הפיקוח חוזר לשורה (14) הלולאה תפסיק להתבצע, כי הפונקציה EOLN(TEXTFILE) החזירה ערך TRUE.

(22) כאשר קוראים לתוך משתנה את הסימן של גבול שורה הוא נחשב כמו תו רווח. אם המשתנה הוא מסוג CHAR הוא יקבל ערך של רווח ואם הוא מסוג מספרי כלשהו, המחשב ימשיך לחפש עד שימצא מספר. בתכנית זו לא רוצים לקרוא את הסימן הזה כדי שלא יתוסף אחד למונה של מספר הרווחים ולכן מבצעים את הפרוצדורה READLN(TEXTFILE), אשר גורמת לכך שהמצביע ידלג עד אחרי גבול השורה. כאשר גבול השורה איננו עוד במאגר, הפונקציה הבוליאנית EOLN(TEXTFILE) שוב תחזור להיות FALSE ונוכל שוב פעם להכנס ללולאה שמתחילה בשורה (14).

אם נפסיק את הלולאה בשורות (14)-(20) במצב של סוף קובץ ולא במצב של סוף שורה, נמצא במאגר את הסימן של סוף קובץ. אם ננסה להמשיך בקריאת הקובץ אחרי ההופעה של סימן זה נגרום לשגיאת ביצוע. במקרה כזה, אין להמשיך בפרוצדורה READLN. הפקודה בשורה (21) דואגת לכך שאם סימן סוף הקובץ נמצא במאגר, ז"א EOF(TEXTFILE) הוא TRUE, לא נבצע את הפרוצדורה READLN(TEXTFILE).

(27) פקודת סגירה של קובץ תמליל אינה סטנדרטית ואינה נחוצה בכל מחשב.

13.2 פרוצדורות לקריאה ולכתיבה

נסקור שנית את הפרוצדורות הקשורות לקריאה וכתיבה. נתחיל ב-READ ו-WRITE.

```
(a)   PROCEDURE READ (VAR K: "קובץ כלשהוא";
                      VAR R: "איבר מתוך הקובץ");
      BEGIN
          R:= K^;
          GET (K)
      END;

(b)   PROCEDURE WRITE (VAR K: "קובץ כלשהוא";
                       VAR R: "איבר מתוך הקובץ");
      BEGIN
          K^:= R;
          PUT (K)
      END;
```

מפעילים פרוצדורות אלו ע"י כתיבה של:

READ (KOVETZ, RESHUMA)

או WRITE (KOVETZ, RESHUMA)

כאשר KOVETZ ו-RESHUMA מוגדרים בתכנית. הקובץ K שבפרוצדורה יכול להיות כל קובץ. אם הוא קובץ TEXT, אז R חייב להיות מסוג CHAR, שהרי איברי הקובץ TEXT הם תמיד מסוג זה. אם קובץ ה-TEXT הוא INPUT או OUTPUT (ז"א יח' קלט/פלט הסטנדרטיות לקריאה או כתיבה), אפשר להשמיט את שם הקובץ ולכתוב READ (CH) או WRITE (CH) בהתאמה, כאשר CH הוא מסוג CHAR.

אפשר גם לקרוא או לכתוב מספר רשומות: R1,R2,...,Rn ולא רק רשומה אחת. לשם כך יש לכתוב את הפרוצדורות בצורה זו:

READ (K,R1,R2,..,Rn) או WRITE (K,R1,R2,..,Rn) בהתאמה.

הקריאה או הכתיבה הינה לפי הסדר של הרשומות. תחילה הרשומה הראשונה, אח"כ הרשומה השנייה, עד הרשומה מספר n.

נובע מכל שנאמר לעיל, שבקבצי תמליל (TEXT), כולל קבצי INPUT ו-OUTPUT, הפרוצדורות READ ו-WRITE משמשות לקריאה וכתיבה של תווים בלבד. עד עתה השתמשנו בפרוצדורות אלו כדי לקרוא ולכתוב נתונים מן הסוגים INTEGER, REAL ו-BOOLEAN ומן הסוג ARRAY OF CHAR בשדות ולא בתווים. לכאורה אי אפשר לקרוא ולכתוב מספרים שלמים וממשיים אם הקבצים INPUT ו-OUTPUT הם קבצי תמליל, אך התשובה היא שהרחיבו את הפרוצדורות כדי שתבצענה פונקציות נוספות, כמו שיפורט בהמשך.

13.2.1 קריאה

נניח ש-T הוא קובץ TEXT. אם T=INPUT אין צורך לכתוב זאת במפורש. נגדיר משתנים M1, M2 וכו' מסוג כלשהו. נכתוב את הפרוצדורה הבאה:

```
READ (T, M1, M2, M3.....)
```

היא תפורש ע"י המהדר בדרך זו:

```
BEGIN
  READ (T, M1);
  READ (T, M2);
  READ (T, M3);
  ......
END
```

אם הסוג של המשתנה Mn הוא CHAR, הקריאה היא של תווים בודדים. אם המשתנה הוא שלם או ממשי, המחשב איננו קורא תו אחר בלבד, אלא ממשיך לקרוא תווים ע"י ביצוע חוזר של GET(T) עד שהוא נתקל בספרה, או בסימן (+ או -) של מספר. הוא ממשיך לקרוא עד שהוא מזהה סוף של מספר כשבמאגר T^ מופיע תו שלא יכול להיות חלק ממספר (למשל אות, רווח, פסיק, גבול שורה, או תו מיוחד).

כל שנאמר לגבי READ תקף גם לגבי READLN, אלא שבתום קריאת המשתנה האחרון, המצביע מתקדם עד אחרי גבול השורה של אותה שורה שבה נמצא המשתנה האחרון.

13.2.2 כתיבה

כתיבה בקבצי תמליל מסוג TEXT נעשית בכמה דרכים.

נניח ש-K1 ו-K2 הם ביטויי כתיבה, אשר אפשר להציגם באחת משלוש תבניות:

(a) B
(b) B:R
(c) B:R:D

B מציין ביטוי אריתמטי, ביטוי בוליאני, תו, או מערך של תווים. מספר המקומות המוקצים להדפסת הערך של ביטוי (הרוחב-R) הוא סטנדרטי לכל סוג של ביטוי. אך אם רוצים לשנות אותו, ניתן לעשות זאת ע"י הוספת המספר R (תבנית b). כדי להדפיס מספרים ממשיים בצורה מקובלת ולא במבנה של מעריך ומנטיסה יש לציין את מספר המקומות D אחרי הנקודה העשרונית (תבנית c).

הפרוצדורה WRITE (T, K1, K2...) תפורש כך ע"י המהדר:

```
BEGIN
  WRITE (T, K1);
  WRITE (T, K2);
  .......
END
```

כשכותבים קובץ OUTPUT אין צורך לרשום את T.
פרוצדורה זו גורמת לכך שהפקודה WRITE תתבצע כמה פעמים שצריך כדי למלא את הדרישות המתחייבות מכל ביטויי הכתיבה, כולל הוספת רווחים בעת הצורך.

כל שנאמר לגבי WRITE תקף גם לגבי WRITELN. לאחר הכתיבה של המשתנה האחרון מדפיסים תו גבול שורה, אשר גורם למדפסת או למסוף לעבור לשורה הבאה ולחזור להמשך הכתיבה. אם כותבים קובץ תמליל בדיסק, נרשמים בו תווי הבקרה האלה.

13.2.3 שיפור התכנית למציאת מלות מפתח

אפשר לשפר את התכנית MILOT_MAFTEACH שבפרק 10 ע"י שימוש בשיטת קבצי תמליל. תכנית זו מצאה את מספרי השורות שבהן מופיעות מלות מפתח שהוזנו כקלט והחסינה את התמליל במערך שהוגדר כך:

```
SHUROT: ARRAY[SUG_MISPAR_SHURA] OF SHURA;
```

החסרונות בשימוש במערך הם שניים:

א. בזבוז מקום: המערך SHUROT תופס 80,000 בתים (1000 שורות של 80 בתים כל אחת) בזיכרון. במחשבים קטנים יש הגבלה של גודל הזיכרון הפנוי.

ב. כמות הנתונים מוגבלת בגודל המערך: בתכנית הכתובה בפרק 10 הוגדר מערך בגודל של 1,000 שורות. לפיכך, התכנית אינה יכולה לטפל בתמליל יותר גדול.

אפשר לעקוף שני חסרונות אלו אם מגדירים קובץ תמליל ומערך של שורה אחת, באופן הבא:

```
VAR
   KOVETZ_TEXT: TEXT;
   SHURA: ARRAY[1..ORECH_SHURA_MAXIMALI] OF CHAR;
```

המערך SHUROT כבר איננו נחוץ. אמנם, מספר השורות בתמליל עדיין קובע את הגודל של המערך HOFAOT, אך זהו מערך בוליאני ולכן אפשר להגדיל אותו בלי לבזבז הרבה מקום בזיכרון. רוב השינויים בתכנית שבפרק 10 מתיחסים לשתי הפרוצדורות הבאות:

TIPUL_BETAMLIL (שורות (129)-(152))
HADPASAT_PELET (MONE_SHUROT) (שורות (85)–(95))

אפשר לכתוב שתי פרוצדורות אלו בצורה שונה:

א. קטע זה מחליף את הפרוצדורה בשורות (129) עד (152):

```
PROCEDURE TIPUL_BETAMLIL;
BEGIN
   WRITELN ('הכנס את הקטע');
   MONE_SHURUT:= 0;
   WHILE NOT EOF DO
      BEGIN
         ORECH_SHURA:= 0;
         MONE_SHUROT:= MONE_SHUROT + 1;
         WHILE NOT EOLN DO
            BEGIN
               READ (TAV);
               WRITE (KOVETZ_TEXT, TAV);
               ORECH_SHURA:= ORECH_SHURA + 1;
               SHURA[ORECH_SHURA]:= OTIYOT_GDOLOT(TAV)
            END;
         READLN;
         WRITELN (KOVETZ_TEXT);
         FOR I:= ORECH_SHURA+1 TO ORECH_SHURA_MAXIMALI DO
            SHURA [I]:= '^';
         IF ORECH_SHURA >= ORECH_HAKATAN_BEYOTER-1 THEN
           MTZA_MILOT_MAFTEACH
```

אין צורך בפרמטר בעבור הפרוצדורה MTZA_MILOT_MAFTEACH כמו *)
שנדרש בנוסח הקודם של התכנית. בנוסח הקודם הפרמטר התייחס
למספר של השורה במערך SHUROT ועכשיו רק מחזיקים את השורה
הנוכחית במערך של שורה אחת, שנקרא SHURA. יש לערוך
בפרוצדורה MTZA_MILOT_MAFTEACH מספר תיקונים נוספים. *)

```
        ELSE
          FOR I:=1 TO MISPAR_MILOT_MAFTEACH DO
            MILOT_MAFTEACH[I].HOFAOT[MONE_SHUROT]:= FALSE
      END
END;
```

ב. קטע זה מחליף את הפרוצדורות בשורות (85) עד (95):

```
PROCEDURE HADPASAT_PELET;          (* גם פה אין צורך בפרמטר *)
VAR                                (* שהיה דרוש בגירסה הראשונה. *)
   I: INTEGER;
   TAV: CHAR;
BEGIN
   I:= 0;
   WHILE NOT EOF DO
      BEGIN
         I:= I+1;
         WRITE (I:3, '^':3);
         WHILE NOT EOLN DO
            BEGIN
               READ (KOVETZ_TEXT, TAV);
               WRITE (TAV)
            END;
         READLN (KOVETZ_TEXT);  (* כדי לעבור לשורה חדשה *)
         WRITELN    (* כדי לעבור לשורה חדשה בהדפסת הפלט *)
      END
END;
```

13.3 עורך תמליל (*)

עורך תמליל (Text Editor), או בקיצור עורך (Editor), הוא תכנית שמאפשרת תיקונים בקבצי תמליל. התמליל שנעסוק בו הוא תמליל באנגלית, כמו למשל תכנית מחשב או כל תמליל אחר.

כדי להדגים את הפונקציות של העורך נציג תכנית אשר מקבלת חמש פקודות עריכה. הכינוי "השורה הנוכחית" בתיאור הפקודות מתייחס לשורה המוצגת במסך.

א. Tn - העבר את n השורות הבאות (כולל השורה הנוכחית) לקובץ החדש.
ב. Mn - מחק את n השורות הבאות (כולל השורה הנוכחית) מן הקובץ החדש.
ג. H - הוסף שורות לפני (L) או אחרי (A) השורה הנוכחית, שהסמן מצביע עליה.

ד.   S - סיים את העריכה. העבר את שאר השורות לקובץ החדש.
ה.   ? - עזרה. הצג הסבר על הפקודות האחרות.

אם רוצים לבנות קובץ חדש, העורך מאפשר למתכנת להזין את תוכן הקובץ. אם הקובץ כבר קיים, העורך פועל כך:
הוא סורק את הקובץ לפי סדר השורות ומציג את השורה הנוכחית, שהסמן מצביע עליה, בראש המסך ואת הפקודות וההודעות הוא מציג בתחתית המסך. המשתמש יכול למחוק את השורה הנוכחית ושורות נוספות כרצונו, הוא יכול להוסיף שורות לפני שורה זו או אחריה והוא יכול לשנות את התוכן של השורה הנוכחית.

בעורך הפשוט המודגם כאן, אפשר לעשות שינוי בשורה הנוכחית רק ע"י מחיקה של השורה והוספה של שורה חדשה במקומה עם התיקונים הרצויים. אם מטפלים בקובץ קיים, העורך מציג אותו במלואו לפני התחלת העריכה והוא מציג אותו תמיד בסיום העריכה.

מן הראוי לציין שתכנית עבור עורך אינטראקטיבי חייבת להיות משולבת במערכת ההפעלה של אותו מחשב ולהתחשב בתכונות המסופים. למשל, בדוגמא זו רשמנו, שאפשר לצאת מן העריכה בכל שלב ע"י הקשת Control-C, אך קיימות מערכות שבהן מפסיקים את העיבוד בדרך שונה. כאן השתמשנו בקוד ANSI (American National Standards Institute) בפרוצדורה להזזת הסמן במסוף, בהתאם למסוף שאנו משתמשים בו. אם משתמשים במסוף מסוג שונה יש להשתמש בקוד מתאים.

תכנית העורך:

```
PROGRAM ORECH(KOVETZ_KAYAM, KOVETZ_CHADASH, INPUT, OUTPUT);
VAR
    KOVETZ_KAYAM, KOVETZ_CHADASH: TEXT;
    SHEM_KOVETZ_KAYAM, SHEM_KOVETZ_CHADASH:
              PACKED ARRAY[1..10] OF CHAR;
    TSHUVA: CHAR;
    MISPAR_HATAVIM: INTEGER;
    SHURA_NOCHCHIT: PACKED ARRAY [1..80] OF CHAR;
    LO_LICHTOV: BOOLEAN;

    PROCEDURE MASACH (Y, X: INTEGER): EXTERNAL;

    PROCEDURE OTEK (VAR K: TEXT);
    VAR
       TAV: CHAR;
       SHURA: 0..MAXINT;
    BEGIN
       RESET (K);
       SHURA:= 1;
       WHILE NOT EOF(K) DO
        BEGIN
          WRITE (SHURA:4, '^^^');
          SHURA:= SHURA+1;
          WHILE NOT EOLN(K) DO
             BEGIN
```

```
                    READ (K, TAV);
                    WRITE (TAV)
                 END;
              READLN (K);
              WRITELN
           END
       END;

        PROCEDURE EZRA;
        BEGIN
           MASACH (6,1);
           WRITELN      ('העורך מקבל את הפקודות הבאות:^^^^^');
           WRITELN;
           WRITELN      ('?   הדפסה של הודעה זו^^^^^^^^^^^^');
           WRITELN      ('ח T  העברה של ח שורות לקובץ החדש^');
           WRITELN      ('ח M  מחיקה של ח שורות^^^^^^^^^^^^^^');
           WRITELN      ('ח H  הוספה של ח שורות^^^^^^^^^^^^^^');
           WRITELN      ('S    סיום העריכה^^^^^^^^^^^^^^^^^^^^');
           MASACH (24,1)
        END;

        PROCEDURE HATZAGAT_SHURA;
        BEGIN
           MASACH (1,1);
           WRITELN     ('השורה הנוכחית^^^^^^^^^^^^^^^^^^^^^^');
           WRITE (SHURA_NOCHCHIT);
           MASACH (24,1)
        END;

        PROCEDURE KTOV_SHURA;                      (* כתיבה *)
        VAR
          I: INTEGER;
        BEGIN
           FOR I:= 1 TO MISPAR_TAVIM DO
              WRITE(KOVETZ_CHADASH, SHURA_NOCHCHIT[I]);
           WRITELN (KOVETZ_CHADASH);
           IF EOF(KOVETZ_KAYAM) THEN
              LO_LICHTOV:= TRUE
        END;

         PROCEDURE KRA_SHURA;                      (* קריאה *)
         VAR
           I, J: INTEGER;
         BEGIN
            J:= 1;
            WHILE NOT EOLN(KOVETZ_KAYAM) DO
               BEGIN
                  READ(KOVETZ_KAYAM, SHURA_NOCHCHIT[J]);
                  J:= J+1
               END;
```

```
            MISPAR_HATAVIM:= J-1;
            FOR I:= J TO 80  DO
               SHURA_NOCHCHIT[I]:= '^';
            READLN (KOVETZ_KAYAM);
            HATZAGAT_SHURA
         END;
         PROCEDURE HATEK_SHURA (VAR K: TEXT);
         VAR
            TAV: CHAR;
         BEGIN
            WHILE NOT EOLN(K)  DO
              BEGIN
                READ (K, TAV);
                WRITE (KOVETZ_CHADASH, TAV)
              END;
            READLN(K);
            WRITELN (KOVETZ_CHADASH)
         END;

         PROCEDURE TIPUL_BEPKUDOT;
         VAR
            MISPAR_SHUROT, I: INTEGER;
            OPERATOR, TSHUVA: CHAR;
            SOF_ARICHA: BOOLEAN;
         BEGIN
           WRITELN;
           WRITELN  ('תוכן של הקובץ הקיים ***');
           OTEK (KOVETZ_KAYAM);
           WRITELN;
           WRITELN;
           WRITELN  ('הקש RETURN כדי להמשיך ***');
           READLN;
           SOF_ARICHA:= FALSE;
           RESET (KOVETZ_KAYAM);
           MASACH (-19, 1);          (* הצבת הסמן ומחיקת מסך *)
           WRITELN  ('העורך מוכן ***');
           WRITELN  ('השורה הנוכחית מוצגת למעלה ***');
           WRITELN ('כדי להפסיק בלי לסיים, לחץ על Control-C');
           KRA_SHURA;
           REPEAT
             MASACH (24, 1);
             WRITE ('בחר באפשרות אחת:');
             WRITE ('T(TAVIR), H(HOSAFA),
                    M(MECHIKA), S(SIYUM), ?(EZRA)');
             READLN (OPERATOR);
             CASE OPERATOR OF
               '?': EZRA;
               'T': BEGIN                          (* העברה *)
                      WRITE ('כמה שורות להעביר?');
                      READLN (MISPAR_SHUROT);
```

```
                    I:= 1;
                    KTOV_SHURA;
                    WHILE NOT EOF (KOVETZ_KAYAM) AND
                                  (I < MISPAR_SHUROT) DO
                       BEGIN
                         HATEK_SHURA (KOVETZ_KAYAM);
                         I:= I+1
                       END;
                    IF NOT EOF (KOVETZ_KAYAM) THEN
                      KRA_SHURA
                    ELSE
                      BEGIN
                        MASACH (-20,1);
                        WRITELN ('סוף הקובץ הקיים');
                        LO_LICHTOV:= TRUE
                      END
                  END;                        (* העברה *)
              'M': BEGIN                       (* מחיקה *)
                     WRITE  ('כמה שורות למחוק?');
                     READLN (MISPAR_SHUROT);
                     I:= 1;
                     WHILE NOT EOF (KOVETZ_KAYAM)
                                AND (I < MISPAR_SHUROT) DO
                        BEGIN
                          READLN (KOVETZ_KAYAM);
                          I:= I+1
                        END;
                     IF NOT EOF (KOVETZ_KAYAM)  THEN
                       KRA_SHURA
                     ELSE
                       BEGIN
                         MASACH (-20, 1);
                         WRITELN ('סוף הקובץ הקיים');
                         LO_LICHTOV:= TRUE
                       END
                   END;                        (* מחיקה *)
              'H': BEGIN                         (* הוספה *)
                     WRITELN ('הוסף שורות כרצונך^^');
                     WRITELN ('לסיום הקש Control-Z');
                     REPEAT
                       WRITELN ('הוסף לפני (L) או אחרי (A)');
                       WRITELN (' השורה הנוכחית^^^^^^^^^^^');
                       READLN (TSHUVA)
                     UNTIL (TSHUVA='A') OR (TSHUVA='L');
                     WRITELN ('התחל להוסיף שורות^^^^^^^^^^');
                     IF (TSHUVA='A') AND NOT LO_LICHTOV THEN
                       KTOV_SHURA;
                     WHILE NOT EOF DO
                       HATEK_SHURA (INPUT);
                     IF (TSHUVA='L') AND NOT LO_LICHTOV THEN
```

```
                    KTOV_SHURA;
                  IF NOT EOF (KOVETZ_KAYAM) THEN
                    KRA_SHURA;
                  RESET (INPUT)
                END;                     (* הוספה *)
           'S': BEGIN                    (* סיום העריכה *)
                  WRITE(SHEM_KOVETZ_CHADASH,'הקובץ החדש:^');
                  IF NOT LO_LICHTOV  THEN
                    KTOV_SHURA;
                  WHILE NOT EOF (KOVETZ_KAYAM)  DO
                    HATEK_SHURA (KOVETZ_KAYAM);
                  SOF_ARICHA:= TRUE
                END                      (* סיום עריכה *)
        END                              (* סיום CASE *)
      UNTIL SOF_ARICHA                   (* REPEAT *)
    END;                                 (* טיפול בפקודות *)

    PROCEDURE  TIPUL_BAKAYAM;
      VAR
        I:  INTEGER;
      BEGIN
        WRITELN  ('מהו השם של הקובץ הקיים?^^^^^^^');
        READLN  (SHEM_KOVETZ_KAYAM);
        RESET (KOVETZ_KAYAM, SHEM_KOVETZ_KAYAM,, I);
        IF  I<O  THEN
           WRITELN  ('אין קובץ בשם זה. נסה שוב')
         ELSE
           TIPUL_BEPKUDOT
      END;

BEGIN                                    (* התכנית הראשית *)
    MASACH  (-19, 1);
    WRITELN   ('מהו שם קובץ הפלט?');
    READLN   (SHEM_KOVETZ_CHADASH);
    REWRITE (KOVETZ_CHADASH, SHEM_KOVETZ_CHADASH);
    REPEAT
        WRITELN ('האם קיים קובץ? ענה כן (K) או לא (L)');
        READLN (TSHUVA)
    UNTIL (TSHUVA='K') OR (TSHUVA='L');
    IF  TSHUVA='L'  THEN
      BEGIN
        WRITELN  ('הוסף שורות כרצונך');
        WRITELN  ('לסיום הקש  Control-Z');
        WHILE NOT EOF DO
          HATEK_SHURA (INPUT)
      END
    ELSE
      TIPUL_BAKAYAM;
    WRITELN;
    WRITELN;
```

```
(217)          WRITELN  ('תוכן של הקובץ החדש ***');
(218)          OTEK (KOVETZ_CHADASH)
(219)     END.
```

<u>הסבר:</u>

(219)-(197) התכנית הראשית.

(198) משתמשים בפרוצדורה למיקום הסמן, כפי שהוסבר בפרק 9, בסעיף על פרוצדורות לשליטה במסוף. תפקידה של הפרוצדורה למקם את הסמן בעמודה הראשונה של שורה 19 במסך. הסימן השלילי מציין שרוצים למחוק את המסך לפני מיקום הסמן.

(201)-(200) שתי שורות אלו מתאימות רק למערכת שבה אפשר לקרוא את השם החיצוני של הקובץ החדש בזמן הביצוע (שורה 200) ולהשתמש בו כמשתנה בפקודת REWRITE (שורה 201).

(205)-(202) בקטע זה מבקשים תשובה לשאלה. אפשר להשיב באות אחת, K או L, אך מותר גם לכתוב מלה שלימה: KEN או LO. מכיון שהמשתנה TSHUVA מוגדר כסוג CHAR, אין קולטים כל מה שנכתב אחרי K או L.

(212)-(206) אם אין קובץ קיים, התכנית מאפשרת רק פעולת הוספה, אשר מבוצעת בצורה איטרטיבית בשורות (210)-(211). עוברים לפרוצדורה HATEK_SHURA (76-87), שקוראת תווים מקובץ תמליל K כלשהו, כאן - קובץ INPUT. הקריאה של כל תו בנפרד נעשית לתוך משתנה בשם TAV (שורה 82), ומיד כותבים אותו בקובץ החדש. כאשר מגיעים לסימן של סוף שורה בקובץ הקלט K, מסיימים את ההעתקה מ-K (80), מקדמים את המצביע עד אחרי תו זה (85) וכותבים סימן סוף שורה בקובץ החדש (86).

(214)-(213) אם המשתמש מבקש לטפל בקובץ קיים קוראים לפרוצדורה TIPUL_BAKAYAM (185-196). בפרוצדורה זו מבצעים פקודה מיוחדת למערכת שבה מריצים את התכנית.

בשורה (191) כוללים בפקודה RESET את השם של הקובץ הקיים שקוראים בשורה (190) וגם את הפרמטר I אחרי שני פסיקים. אם מנסים לעשות RESET לקובץ שאיננו קיים, מקבלים בזמן הביצוע הודעת שגיאה. המשתנה I מורה למערכת ההפעלה לא לשלוח הודעת שגיאה, אלא לתת ל-I ערך שלילי ולהפסיק בכך את הביצוע של התכנית. ע"י כך אפשר בשורה (193) להודיע שאין קובץ קיים ובכך לסיים את התכנית. אם הקובץ קיים עוברים לפרוצדורה TIPUL_BEPKUDOT (88-184), כדי לתרגם את פקודות העורך.

<u>(184)-(88) הפרוצדורה TIPUL BEPKUDOT:</u>

פרוצדורה זו בנויה מחלקים אחדים, בתוכם גם פרוצדורות אחרות. מתחילים בביצוע הפרוצדורה OTEK (11-30). היא דומה מאוד לפרוצדורה HATEK_SHURA שתיארנו קודם, עם השינויים האלה:

א. היא מעבירה מקובץ תמליל K למסוף ולא ל-KOVETZ_CHADASH.
ב. בזמן הכתיבה היא מוסיפה מספרי שורה.
ג. היא מעתיקה את כל הקובץ ולא רק שורה אחת.

הנתונים מהקובץ הקיים מוצגים על המסך לעיון המשתמש. להמשיך, הוא חייב להקיש RETURN (זוהי תוצאת הפקודה READLN בשורה (100)). אח"כ פותחים את הקובץ הקיים (102), שנסגר לאחר שנקרא עד סופו בפרוצדורה OTEK.

בשלב זה קוראים את השורה הראשונה בקובץ הקיים ומציגים אותה במסך (107). הפרוצדורה שעושה פעולה זו (60-75) קוראת מן הקובץ עד סוף השורה ושומרת במשתנה MISPAR__HATAVIM (70) את מספר התווים בשורה. משתנה זה הוא משתנה גלובלי, כי נצטרך אותו בזמן כתיבת השורה לתוך הקובץ החדש.

השורה המוצגת על המסך (SHURA_NOCHCHIT) היא בת 80 מקומות. כותבים במקומות הריקים תווי רווח (71-72) כדי שלא ישארו בהם תווים מן השורה הקודמת. הפרוצדורה הזו קוראת לפרוצדורה HATZAGAT_SHURA, שמציגה את השורה במסך (שורה 74).

(108-183) לולאה זו מטפלת בכל פקודה בנפרד. מציגים את האות הראשונה בכל פקודה (לפי הפירוט שבשורות 111-112) ולפי תשובת המשתמש נקבע מסלול ביצוע התכנית.

(115) אם התשובה היא ? (סימן שאלה) מסתעפים לפרוצדורה EZRA (31-42) שם מתוארות הפקודות השונות.

(116) אם התשובה היא T (העברה) שואלים את המשתמש כמה שורות הוא רוצה להעביר (117) ומעבירים בהתאם.

תחילה מעבירים את השורה שנמצאת עכשיו ב-SHURA_NOCHCHIT. העברה זו נעשית ע"י הפרוצדורה KTOV_SHURA (50-59). היא מנצלת את המשתנה הגלובלי MISPAR_TAVIM שמכיל את מספר התווים בשורה SHURA_NOCHCHIT, כדי לדעת כמה תווים לכתוב בקובץ החדש. אם השורה שמעבירים מ-SHURA_NOCHCHIT אל KOVETZ_CHADASH היא השורה האחרונה בקובץ, יש למנוע את הפעולות הבאות, אשר יוצרות מצבי שגיאה:

א. קריאה של שורה נוספת מן הקובץ הקיים לתוך SHURA_NOCHCHIT. כדי למנוע זאת בודקים בשורה (127) אם נמצאים במצב של EOF(KOVETZ_KAYAM) לפני שמבצעים את הפרוצדורה KRA_SHURA.

ב. כתיבה של השורה שנמצאת ב-SHURA_NOCHCHIT כאשר נמצאים בסיום התכנית לתוך הקובץ החדש על אף שכבר כתבנו אותה. קשה להגן מפני שגיאה זו כי גם אם EOF(KOVETZ_KAYAM) הוא TRUE, יתכן שהעברנו את השורה האחרונה מן הקובץ הקיים ל-SHURA_NOCHCHIT, אבל עוד לא כתבנו אותה בקובץ החדש ואז צריך לכתוב אותה בסיום התכנית. הפתרון הוא לתת ערך TRUE למשתנה הבוליאני LO_LICHTOV אחרי שמעבירים את השורה האחרונה בקובץ הקיים מ-SHURA_NOCHCHIT אל KOVETZ_CHADASH (שורה (58)-(57)), או אחרי שמעתיקים את השורה האחרונה מן הקובץ הקיים אל הקובץ החדש (שורה 133). לא נותנים ערך TRUE למשתנה LO_LICHTOV מיד עם קריאת השורה האחרונה (בפרוצדורה KRA_SHURA).

(126)-(119) המשתמש מבקש להעביר n שורות. מעבירים תחילה את השורה הנוכחית (120) ואח"כ את n-1 השורות הנוספות מן הקובץ הקיים ישר אל הקובץ החדש (126)-(121).

(127)-(128) לאחר שמעבירים לקובץ החדש את השורה שהופיעה במסך עם n-1 שורות נוספות, צריך להציג בו את השורה הבאה לאחר שורות אלו, כל עוד יש שורות בקובץ.

(132)-(133) אם לא נשארו שורות בקובץ הקיים מודיעים זאת למשתמש (132). כזכור, הופכים את LO_LICHTOV ל-TRUE כדי שלא ננסה להעתיק פעם נוספת את התוכן של SHURA_NOCHCHIT לקובץ החדש (133).

(136)-(154) כדי למחוק n שורות צריך לבצע שתי פעולות:
א. לא להעביר אל הקובץ החדש את השורה שמוצגת במסך.
ב. לקרוא n-1 שורות מהקובץ הקיים ולא לכתוב אותן (140-145) בקובץ החדש.

(146)-(147) במקרה טיפוסי צריך להציג במסך את השורה הבאה מהקובץ הקיים.

(148)-(153) אם מוחקים שורות עד לסוף הקובץ הקיים, צריך להודיע זאת למשתמש (151). כדי שהתכנית תדע עובדה זו ולא תנסה לכתוב אח"כ מתוך SHURA_NOCHCHIT, משווים את המשתנה הבוליאני LO_LICHTOV ל-TRUE (שורה (152)).

(155)-(173) בפעולת הוספה לא צריך לדעת כמה שורות המשתמש רוצה להוסיף, אך חייבים לדעת אם הוא רוצה להוסיף אותן לפני או אחרי השורה הנוכחית (158-162). אם הוא רוצה להוסיף את השורות החדשות אחרי השורה הנוכחית, כותבים את השורה הנוכחית לתוך הקובץ החדש מיד (164–165). בכל מקרה, חייבים להעתיק שורות מן המסך לתוך הקובץ החדש (166-167). אם מכניסים את השורות החדשות לפני השורה שהיתה שם קודם, צריך לכתוב את השורה הנוכחית לתוך הקובץ החדש (168-169).

אם לפני הוספת השורות לא נמצאים במצב של EOF, מציגים שורה חדשה במסך (170-171). בכל מקרה, מצב חדש של EOF בקובץ הקיים לא יכול לקרות כתוצאה מהוספת שורות. אבל, כתוצאה מההוספה נוצר מצב EOF בקובץ INPUT (מלוח המקשים), כי מסיימים את הכנסת השורות ע"י הקשה של Control-Z שמפעיל את EOF(INPUT). לשם כך מכבים מתג זה בפקודה RESET (172).

(174)-(181) בזמן סיום הטיפול בקובץ כותבים את השורה שנמצאת ב-SHURA_NOCHCHIT אל הקובץ החדש אם לא כתבנו אותו כבר (176-177) ומעבירים את שאר השורות מהקובץ הקיים (178-179). לאחר מכן מדליקים את המתג שמציין את סיום העריכה (180).

(183) סיום רגיל של התכנית (לעומת סיום כפוי ע"י Control-C) נעשה רק כאשר המתג SOF_ARICHA נדלק. בסיום התכנית מציגים במסך את הקובץ החדש (217-218) כדי שהמשתמש יוכל לעיין בו ולבדוק את השינויים שערך.

13.4 תרגילים

13.4.1 שאלות

(1) (*) איך תשנה את העורך שהודגם כדי שהוא יאפשר פקודה חדשה לביצוע החלפה של מחרוזות תווים.

מבנה הפקודה: S/MACHROZET1/MACHROZET2

הפעולה היא החלפה של התווים אשר במחרוזת MACHROZET1 שבשורה הנוכחית, בתווים אשר במחרוזת MACHROZET2.

(2) (*) עליך לשנות את התכנית עבור עורך כדי שיהיה אפשר לנוע בין שורות התמליל לפנים ולאחור. אפשר להכניס את הקובץ הקיים לתוך מערך ולהציג בכל פעם רק 15 שורות תמליל כשבראש כל אחת ניתן מספר השורה. את פקודות העורך אפשר להציג בתחתית המסך.

השתמש בפקודות העורך הבאות:

F n	התקדם n שורות.
B n	נוע לאחור n שורות.
D n1:n2	מחק משורה n1 עד שורה n2 (כולל).
A {B} n {A} n	הוסף n שורות לפני ("B"), או אחרי ("A") השורה הנוכחית.
C/MACHROZET1/MACHROZET2/n1:n2	החלף MACHROZET1 ב-MACHROZET2 משורה n1 עד שורה n2.
E	סיום.

הפקודה T כבר אינה נחוצה, כי כל הקובץ מוחזק במערך בעת העיבוד ומועבר כולו לקובץ החדש בסיום העריכה.

13.4.2 תכניות

(1) כתוב תכנית שתטעין את המשפטים הבאים לתוך קובץ. שים רווח אחד בסוף כל משפט ושני רווחים לפני כל פיסקה חדשה.

THE AUTOMATIC ELECTRONIC DIGITAL COMPUTER IS A MACHINE THAT USES
ELECTRONIC CIRCUITS TO MANIPULATE DATA EXPRESSED IN A SYMBOLIC
FORM ACCORDING TO SPECIFIC RULES IN A PREDETERMINED WAY. LET'S
EXPLAIN THIS IN DETAIL.
FIRST OF ALL THE COMPUTER IS A MACHINE. THIS MEANS IT CAN PERFORM
ONLY THOSE ACTIVITIES FOR WHICH THE BASIC CAPABILITIES HAVE BEEN
SPECIFICALLY DESIGNED INTO THE MACHINE.
SECONDLY, THE COMPUTER IS AUTOMATIC. THIS MEANS THAT ONCE STARTED,
IT CONTINUES TO RUN WITHOUT OUTSIDE INTERFERENCE.
THIRDLY, IT IS ELECTRONIC: THAT IS, IT IS MADE UP OF ELECTRONIC
CIRCUITS AND RUNS ON ELECTRICAL ENERGY.
FOURTHLY, THE COMPUTER IS A SYMBOL MANIPULATOR. IT MANIPULATES
DATA, NOT PHYSICAL ENERGY.
FIFTH, THE COMPUTER CAN PERFORM ONLY THE PROCESSES OF ADDITION,
SUBTRACTION, MULTIPLICATION, DIVISION AND COMPARISON, IN ADDITION
TO DATA TRANSFER BETWEEN COMPUTERS.

SIXTH, SOMEONE (THE PROGRAMMER) MUST PREPARE A FINITE SEQUENCE OF THE ALLOWABLE INDIVIDUAL OPERATIONS (A PROGRAM) FOR THE COMPUTER TO FOLLOW.
FINALLY, THE COMPUTER CAN STORE THE PROGRAM WITHIN ITS OWN MEMORY AND THEN FOLLOW IT THROUGH UNDER ITS OWN DIRECTION WITHOUT FURTHER GUIDANCE.

(2) כתוב תכנית שתקרא כקלט את הקובץ שבנית בשאלה (1) ותדפיס אותו בשורות שאורכן N מקומות. את אורך השורה יש לקבל כקלט. המשפט הראשון בכל פיסקה צריך להתחיל במקום החמישי בכל שורה. אסור לחלק מלה בין שתי שורות.

(3) כתוב תכנית שתחפש מלים בקובץ ותדווח עליהם. התכנית תקבל מלה אחת, למשל COMPUTER ותדפיס:

א. את מספר הפעמים שמלה זו מופיעה בקטע.

ב. את 20 התווים שקודמים למלה זו ואת 20 התווים שאחריה. השתמש כקלט בפיסקה שבשאלה (1).

(4) (*) כתוב תכנית לשאלה (7) שבפרק 11 והשתמש בקבצי תמליל. בדרך זו תימנע מהגדרת מערך באורך של 20 לכל מלה שתופיע באינדקס.

(5) כתוב תכנית שמוצאת בקובץ תמליל כלשהו, את:-

- מספר השורות.
- אורך מינימלי של שורה.
- אורך מקסימלי של שורה.
- אורך ממוצע של שורה.

(6) (*) כתוב תכנית שמקבלת כקלט:

- שם קובץ המורכב מרשומות.
- משתנים שלמים A ו-B.

התכנית צריכה לבדוק אם הקובץ ממויין לפי השדה שמתחיל בעמודה A ומסתיים בעמודה B. אפשר להניח שנתון: B-A<20.

(7) (*) כתוב תכנית להכנת פלט הדפסה (Formatter) של קובץ תמליל באנגלית. התכנית מאפשרת להקים קובץ תמליל ולהוסיף בו הוראות שתתורגמנה בזמן ההדפסה ע"י מערכת ההפעלה להוראות למדפסת. ההוראות תכתבנה תמיד בראש שורה נפרדת, לפני השורה שהן תפעלנה עליה.

- דלג על n שורות בזמן ההדפסה *Sn
- התחל פיסקה חדשה והשאר n רווחים *Pn
- מרכז את השורה הבאה שהיא כעין כותרת *C

בתחילת קובץ התמליל יכתבו שני משתנים A ו-B, שמשמעותם היא:

A: הרוחב של כל שורה הוא עד A מקומות. אסור לקצץ מלה ולכן לפעמים הרוחב יהיה פחות מ-A.

B: דלג B שורות בין שורה אחת לחברתה. דלג 2B שורות בין פיסקה לפיסקה.

פרק 14
קבוצות

14.1 הגדרה של קבוצה

קבוצה (Set) מוגדרת במתמטיקה כאוסף של עצמים (Objects). באות A למשל, נסמל קבוצת תלמידים בכיתה ובאות B נסמל קבוצת חפצים שנמצאים בחדר. העצמים שמרכיבים את הקבוצה נקראים גם אלמנטים (Elements), או איברי הקבוצה. הגדרה של קבוצה בפסקל מוגבלת יותר מהגדרתה במתמטיקה, כפי שנראה להלן:

א. איברי הקבוצה חייבים להיות מאותו סוג. למשל, הם יכולים להיות תווים מסוג CHAR או מספרים שלמים, אבל הם לא יכולים להיות משני הסוגים. סוג האיברים נקרא סוג הבסיס (Base Type).

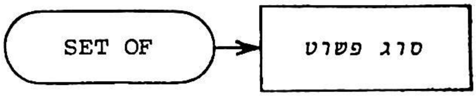
ב. איברי קבוצה בפסקל צריכים להיות סודרים (Ordinal) וניתנים למיון בסדר עולה. דרישה זו מוציאה מן הכלל מערכים ורשומות, שהרי אי אפשר לקבוע ביחס למערך או רשומה אם הוא קטן, שווה או גדול מחברו. נתייחס למשל, למערכים דו-מימדיים A ו-B. נניח: A[1,1]<B[1,1] וגם A[1,2]>B[1,2]. במקרה כזה אי אפשר לומר מהו היחס בין מערך A למערך B: האם A קטן מ-B, גדול מ-B או שווה לו. מצב דומה יכול לקרות ביחס לשדות מקבילים ברשומות A ו-B.

אפשר לסכם דרישות (א) ו-(ב) ולומר שאיברי קבוצת פסקל חייבים להיות סודרים ומאותו סוג. שתי דרישות אלו יחד מבטלות את האפשרות שקבוצה בפסקל תכלול את אותו האיבר פעמים אחדות, שהרי בסדר מיון מסוים, איבר כלשהוא לא יכול להופיע יותר מפעם אחת.

ג. איברי קבוצת פסקל לא יכולים להיות מן הסוג REAL. הסיבה לדרישה זו היא שבזיכרון המחשב מוקדשת סיבית (Bit) אחת לכל איבר פוטנציאלי בקבוצה, בשעה שגודלה של קבוצת המספרים הממשיים שיכולים להיות מיוצגים במחשב הוא אינסופי. אי אפשר להקדיש מספר סופי של סיביות בכדי לייצג קבוצה זו ולכן גם אי אפשר להגדיר אותה בפסקל.

14.2 הצהרה ובנייה של קבוצות

כדי ליצור קבוצה יש להצהיר תחילה על סוג האיברים, אשר כלולים בה:



סוג איברים פשוט הוא <u>סוג הבסיס</u>. הוא מזהה את כלל איברי הקבוצה באחת מ-3 דרכים:

א. התייחסות לסוג המוגדר מראש: למשל, CHAR או סוג שהוגדר בהצהרת TYPE.
ב. מנייה: פירוט של כל איברי הקבוצה.
ג. קביעת תת-תחום של סוג אשר כבר מוגדר.

הצהרת הסוג יכולה להופיע בהצהרת TYPE או בהצהרת VAR מייד לאחר הנקודתיים שרשומים אחרי שם הקבוצה.

נציג דוגמאות תכנות אחדות כדי להסביר את הגדרת הקבוצה.

<u>דוגמא ראשונה (צבעים):</u>

```
TYPE
   TZEVA_RISHONI= (ADOM, TZAHOV, KACHOL);
   KVUTZAT_TZEVAIM= SET OF TZEVA_RISHONI;
VAR
   TZEVAIM: KVUTZAT_TZEVAIM;
```

בדוגמא הזו, המשתנה הקבוצתי TZEVAIM הוגדר בשלושה שלבים: תחילה מצהירים על הסוג שנקרא TZEVA_RISHONI, אח"כ - על KVUTZAT_TZEVAIM כקבוצה המורכבת מן האיברים אשר מרכיבים את TZEVA_RISHONI. לבסוף מצהירים על המשתנה TZEVAIM כמשתנה קבוצתי, שיכול לקבל בכל עת אחד מן הערכים האפשריים של KVUTZAT_TZEVAIM.

מה הם הערכים האפשריים של KVUTZAT_TZEVAIM? נתאר קבוצה ע"י פירוט איבריה בתוך סוגריים מרובעים. רשימת הקבוצות האפשריות המורכבות מן האיברים של TZEVA_RISHONI רשומה למטה. אם וכאשר יש ערך למשתנה הקבוצתי TZEVAIM, ערכו חייב להיות שווה לאחת מן הקבוצות האלו.

<u>קבוצות בנות שלושה איברים</u>
[ADOM, TZAHOV, KACHOL]

<u>קבוצות בנות שני איברים</u>
[ADOM, TZAHOV]
[ADOM, KACHOL]
[TZAHOV, KACHOL]

<u>קבוצות בנות איבר אחד</u>
[ADOM]
[TZAHOV]
[KACHOL]

<u>קבוצות בנות 0 איברים</u>
[ˆ]

קבוצה בת 0 איברים נקראת קבוצה ריקה (Empty Set). קבוצה זו מהווה אחד מן הערכים האפשריים של כל משתנה קבוצתי.

הסדר לפיו רשומים איברי הקבוצה איננו חשוב, כי הקבוצה הינה אוסף בלתי מסודר של איברים אשר ניתנים לסידור. לכן נוכל לרשום את השיוויונות הבאים:

[KACHOL,ADOM,TZAHOV] = [ADOM,TZAHOV,KACHOL] = [KACHOL,TZAHOV,ADOM] = [TZAHOV,KACHOL,ADOM] = [TZAHOV,ADOM,KACHOL] = [ADOM,KACHOL,TZAHOV]

רשימת כל הקבוצות שאפשר להרכיב מסוג הבסיס היא קבוצת העוצמה (Power Set) של סוג הבסיס. אם יש N איברים בסוג הבסיס, גודלה של קבוצת העוצמה שווה תמיד ל-2^N. בדוגמא זו N שווה ל-3 ולכן גודלה של קבוצת העוצמה הוא $2^3=8$.

הצהרת משתנה בקבוצת TZEVAIM יכולה להיעשות בשני שלבים, או אף בשלב אחד, במקום בשלושה שלבים הרשומים למעלה. ההצהרה בשניים או בשלושה שלבים מאפשרת להגדיר כמה משתנים על סמך מה שהוגדר ב-TYPE.

הצהרה בשני שלבים:

```
TYPE
    TZEVA_RISHONI= (ADOM, TZAHOV, KACHOL);
VAR
    TZEVAIM: SET OF TZEVA_RISHONI
```

הצהרה בשלב אחד:

```
VAR
   TZEVAIM: SET OF (ADOM, TZAHOV, KACHOL);
```

דוגמא שנייה (אותיות):

```
TYPE
    OTIYOT= 'A'..'Z';
    KEVUTZAT_OTIYOT= SET OF OTIYOT;
VAR
    SHEM, KTOVET: KVUTZAT_OTIYOT;
```

בדוגמא זו, הקבוצה OTIYOT הוגדרה כתת-תחום של התווים מסוג CHAR. גם כאן אפשר היה לקצר את הגדרת המשתנים הקבוצתיים SHEM ו-KTOVET לשני שלבים או אפילו לשלב אחד. המשתנים SHEM ו-KTOVET מייצגים את קבוצת האותיות שמשתתפות בשם מסוים ובכתובת מסוימת בהתאמה.

בנייה של קבוצות

בניית קבוצה הינה הצבת ערך למשתנה קבוצתי. את איברי הקבוצה רושמים בתוך סוגריים מרובעים בצד ימין של פקודת ההשמה. אם חלק מאיברי הקבוצה שייכים לתת-תחום של סוג הבסיס, מ-X עד Y למשל, אפשר להשתמש בסימון X..Y.

דוגמא:

```
TYPE
     YOM= (RISHON,SHENY,SHLISHI,RIVIEY,CHAMISHI,SHISHI,SHABAT);
     KEVUTZAT_YAMIM= SET OF YOM;
```

```
VAR
    YEMAI_CHOL, SOF_SHAVUA, YEMAI_MISRAD: KEVUTZAT_YAMIM;
```

פקודות הצבה של ערכים התחלתיים למשתנים הקבוצתיים:

```
 (I)  YEMAI_CHOL:= [RISHON,SHENI,SHLISHI,RIVIEY,CHAMISHI];

(II)  SOF_SHAVUA:= [SHISHI, SHABAT];

(III) YEMAI_MISRAD:= [SHENI, SUCC(SHENI)];
```

ניתן לקצר את פקודת ההצבה (I) ולכתוב:

```
YEMAI_CHOL:= [RISHON...CHAMISHI]
```

גם את פקודה (II) ניתן לכתוב כתת-תחום, אבל אין חיסכון במקרה הזה.

פקודה (III) מדגימה את העובדה שאיברי הקבוצה יכולים להופיע כביטויים אריתמטיים או כפונקציות, כל עוד ערכם הסופי שייך לסוג הבסיס. דוגמא נוספת לכך נראה בהמשך:

```
VAR
   OTIYOT: SET OF 'A'..'Z';
   MISPAR_AKRAI: 0..25;
BEGIN
   READ (MISPAR_AKRAI);
   OTIYOT:= [CHR(ORD('A') + MISPAR_AKRAI)];
END;
```

אם עובדים על מחשב שפועל בקוד ASCII, סכום הקוד של 'A' (כלומר (ORD('A')) ומספר אקראי בין 0 ל-25 יתן את הקוד של אות כלשהיא בין 'A' לבין 'Z'. כאשר מפעילים את פונקצית CHR על אותו קוד, הקוד הדצימלי יוחלף באות המוצגת על ידו והקבוצה OTIYOT תכיל את אותה אות.

דוגמא:

```
VAR
    K: SET OF 1..50;
    :
BEGIN
    K:= [I+1..J, 12, I*J ]
```

K מכיל את הערך 12 אם J=3, I=4.
K מכיל את הערכים 12, 15 אם J=3, I=5.
K מכיל את הערכים 4, 5, 12, 15 אם J=5, I=3.

14.3 השוואה של קבוצות

14.3.1 אופרטורים יחסיים וקבוצות חלקיות

בעזרת אופרטורים (Operators) מתאימים ניתן לבצע את הפעולות הבאות:

א. השוואה בין שתי קבוצות בכדי לבדוק אם הן זהות.
ב. בדיקה אם קבוצה אחת מוכלת בתוך קבוצה אחרת.
ג. בדיקה אם אלמנט כלשהוא שייך לקבוצה מסוימת.

טבלת האופרטורים ניתנת להלן:

אופרטור		פ ע ו ל ה
(1)	=	זהות בין קבוצות
(2)	<>	אי שיוויון בין קבוצות
(3)	<=	הכלת הקבוצה משמאל לסימן בתוך הקבוצה מימין לו
(4)	>=	הכלת הקבוצה מימין לסימן בתוך הקבוצה משמאל לו
(5)	IN	שייכות לקבוצה

<u>הסבר:</u>

א. התוצאה של הפעלת האופרטורים שמופיעים בטבלה היא ערך בוליאני.

ב. הביטוי A<=B מתייחס להכלה של A בתוך B, וערכו של הביטוי הבוליאני הוא TRUE אך ורק אם כל איבר ב-A שייך גם ל-B. כאשר תנאי זה מתקיים, A נקרא (במינוח מתמטי) <u>קבוצה חלקית</u> (Subset) של B. בתורת הקבוצות, יחס זה מסומן כך: $A \subset B$.

ג. הביטוי A>=B מתייחס להכלה של B בתוך A, וערכו של הביטוי הבוליאני הוא TRUE אך ורק אם כל איבר ב-B שייך גם ל-A. כאשר תנאי זה מתקיים B נקרא קבוצה חלקית של A.

ד. שיוויון, או זהות, בין הקבוצות A ו-B קיים כאשר A<=B וגם A>=B. כלומר, כל איבר של A נמצא גם ב-B ולהיפך.

ה. כאשר משתמשים באופרטור IN, האופרנד השמאלי חייב להיות ערך סודר (אורדינלי) השייך לסוג הבסיס, שנקרא לו T. האופרנד הימני חייב להיות קבוצה שהסוג שלה הוא: SET OF T. הערך הבוליאני של התנאי הוא TRUE אם הערך הסודר בצד שמאל שייך לקבוצה שבצד ימין, אחרת הערך הבוליאני של התנאי הוא FALSE.

לדוגמא, הערך של התנאי שמופיע למטה הוא TRUE:

```
A:= 7
A  IN  [1, 3..5, 7]
```

אפשר גם לכתוב [A]<=[1,3..5,7], אבל הייצוג הקודם של התנאי הוא יותר ברור ופשוט. אם רוצים לבדוק בו בזמן את השייכות של A וגם של B לקבוצה, כאשר A<>B, יש לנסח את התנאי באחת מתוך שתי הצורות הבאות:

(a) [A, B] <= [1,3..5,7] (b) A IN [1,3..5,7] AND B IN [1,3..5,7]

במקרה הזה, הייצוג הראשון יותר פשוט וברור.

הקבוצה [1,3..5,7] יכולה להופיע בתכנית בלי הצהרה מראש ואז היא נחשבת <u>כקבוע קבוצתי</u> (Set Constant). ההצהרה נחוצה רק עבור משתנה שבמשך התכנית יוכל לייצג קבוצה בעזרת פקודת השמה, כמו למשל A:=[1,3..5,7].

נניח שקיימת הצהרת VAR הבאה:

```
VAR
   K1,K2,K3,K4,K5,K6: SET OF 1..20;
```

בתוך התכנית בונים בעזרת פקודות הצבה את הקבוצות האלו:

שם הקבוצה	תוכן הקבוצה	תיאור הקבוצה
K1	[1..20]	שלמים עד 20
K2	[2,4,6,8,10,12,14,16,18,20]	שלמים זוגיים, עד 20
K3	[3,6,9,12,15,18]	כפולות של 3, עד 20
K4	[4,8,12,16,20]	כפולות של 4, עד 20
K5	[5,10,15,20]	כפולות של 5, עד 20
K6	[6,12,18]	כפולות של 6, עד 20

<u>14.3.2 יחסים בין קבוצות</u>

ננתח יחסים שונים בין הקבוצות שהוגדרו ונשתמש בטבלת היחסים שהוצגה בסעיף 14.3.1. מספרי הסעיפים מותאמים למיספור האופרטורים בטבלת היחסים.

<u>אופרטור 1 - זהות בין קבוצות:</u>

א. הערך של התנאי K1=[1..10,19,12,7..20] הוא TRUE. בצד ימין של ביטוי התנאי אפשר לרשום ערכים ללא סדר כלשהו, או לרשום ערך כלשהו יותר מפעם אחת. אין לכך השפעה על התוצאה, מכיון שהדבר החשוב הוא, שכל ערך בין 1 ל-20 נמצא לפחות פעם אחת בין הסוגריים המרובעים ולא מופיע אף ערך אחד מחוץ לגבולות אלה. לכן, הקבוצה בצד ימין באמת זהה ל-K1.

ב. הערך של התנאי K1=[20..1] הוא FALSE. תת-הטווח 20..1 נחשב ריק, מפני שהגבול התחתון (20) גדול מן הגבול העליון.

<u>אופרטור 2 - אי-שוויון בין קבוצות:</u>

א. התנאי K6<>[0,6,12,18] הוא TRUE, כי המספר 0 לא נכלל בסוג הבסיס ולכן הוא אינו יכול להיות ב-K6.

ב. הערך של הביטוי K1<>[9..20,11,3..11,1,2] הוא FALSE. הקבוצה K1 תהיה זהה לקבוצה בצד ימין, לאחר שנוריד את הכפילויות ולא נתייחס לסדר בקבוצה שבצד ימין לסימן היחס.

<u>אופרטורים 3, 4 - הכלה של קבוצה אחת בקבוצה אחרת:</u>

א. הערך של התנאי K5<=K1 הוא TRUE. הכפולות של 5 (מ-5 עד 20) מהוות קבוצה חלקית לקבוצת השלמים (מ-1 עד 20).

ב. התנאי K1<=K2 הוא FALSE. הקבוצה K1 מכילה את כל השלמים מ-1 עד 20 כאשר הקבוצה K2 מכילה רק את המספרים הזוגיים בקבוצה זו. התנאי ההופכי K1>=K2 הוא TRUE, כלומר אמת.

ג. התנאי K1<=[ˆ] הוא FALSE. יש בקבוצה K1 איברים שאינם נמצאים בקבוצה הריקה. אם יש ב-K1 אפילו איבר אחד בלבד, קבוצה זו אינה ריקה והתנאי אינו מתקיים. מאידך, התנאי [ˆ]<=K הוא TRUE עבור כל קבוצה K, מכיוון שכל איבר שקיים בקבוצה הריקה (למעשה אין בה אף איבר אחד), נמצא גם ב-K.

<u>אופרטור 5 - שייכות לקבוצה:</u>

אם המשתנה השלם A שווה 1, הערך של הביטוי הבוליאני "A IN K1" הוא TRUE והערך של הביטוי הבוליאני "A IN K2" הוא FALSE.

אם I, J, K, N הם משתנים מאותו סוג סודר ורוצים לכתוב את הביטוי הזה:

```
(I = N)  AND  (J = N)  AND  (K = N)
```

אפשר לכתוב אותו בצורה מקוצרת:

```
[I,J,K] = [N]
```

את התנאי הזה

```
(I = N)  OR  (J = N)  OR  (K = N)
```

אפשר לכתוב כך:

```
N  IN  [I, J, K]
```

<u>14.3.3 פעולות בקבוצות</u>

בתכנית הבאה נדגים יישום המבוסס על השוואה בין קבוצות. התכנית מציגה אלגוריתם כדי למצוא את המלים בנות 3 אותיות שאפשר להרכיב מן האותיות של המלה SHALOM.

```
PROGRAM PIRUK_MILA (OUTPUT);
CONST
        OT_RISHONA = 'A';
        OT_ACHRONA = 'S';
        REVACH = ' ';
        MILIM_BESHURA = 15;
TYPE
        OTIYOT = OT_RISHONA..OT_ACHRONA;
VAR
        OT1, OT2, OT3: OTIYOT;
        MILA, MILA_LEPIRUK: SET OF OTIYOT;
```

```
(12)          MISPAR_MILIM: 1..MILIM_BESHURA;
(13)     BEGIN
(14)          MILA_LEPIRUK:= ['S','H','A','L','O','M'];
(15)          MISPAR_MILIM:=1;
(16)          FOR OT1:= OT_RISHONA  TO  OT_ACHRONA  DO
(17)              FOR  OT2:= OT_RISHONA  TO  OT_ACHRONA  DO
(18)                 IF  OT1 <> OT2  THEN
(19)                   FOR OT3:=OT_RISHONA  TO OT_ACHRONA DO
(20)                     IF  NOT  (OT3 IN [OT1,OT2])  THEN
(21)                       BEGIN
(22)                         MILA:= [OT1, OT2, OT3];
(23)                         IF (MILA <= MILA_LEPIRUK)  THEN
(24)                           BEGIN
(25)                             WRITE (OT1, OT2, OT3, REVACH);
(26)                             IF MISPAR_MILIM= MILIM_BESHURA THEN
(27)                                BEGIN
(28)                                  WRITELN;
(29)                                  MILIM_BESHURA:=1
(30)                                END
(31)                             ELSE
(32)                                MISPAR_MILIM:= MISPAR_MILIM+1
(33)                           END
(34)                       END
(35)     END.
```

<u>הסבר:</u>

בתכנית זו עוברים באופן שיטתי על כל השלישיות שאפשר להרכיב מאותיות האלפבית הלטיני מ-'A' עד 'S' וקובעים אם כל אחת מהן מוכלת בתוך המלה SHALOM. אם התשובה חיובית לגבי שלישיה מסוימת, התכנית מדפיסה אותה. בדרך זו מוצאים את כל התמורות בנות 3 אותיות מתוך 6 האותיות אשר מרכיבות את המלה הנבדקת: S,H,A,L,O,M. מספר התמורות האפשרי הוא 120 (6*5*4=120), כמו שנוכל לראות מן הפלט של התכנית:

```
AHL AHM AHO AHS ALH ALM ALO ALS AMH AML AMO AMS AOH AOL AOM
AOS ASH ASL ASM ASO HAL HAM HAO HAS HLA HLM HLO HLS HMA HML
HMO HMS HOA HOL HOM HOS HSA HSL HSM HSO LAH LAM LAO LAS LHA
LHM LHO LHS LMA LMH LMO LMS LOA LOH LOM LOS LSA LSH LSM LSO
MAH MAL MAO MAS MHA MHL MHO MHS MLA MLH MLO MLS MOA MOH MOL
MOS MSA MSH MSL MSO OAH OAL OAM OAS OHA OHL OHM OHS OLA OLH
OLM OLS OMA OMH OML OMS OSA OSH OSL OSM SAH SAL SAM SAO SHA
SHL SHM SHO SLA SLH SLM SLO SMA SMH SML SMO SOA SOH SOL SOM
```

(1) השם של התכנית PIRUK_MILA מורה על כך שבתכנית זו נפרק מלה בצורות שונות.

(4)-(3) נותנים שמות לאות הראשונה ולאות האחרונה מתוך האלפבית הלטיני, אשר נמצאות במלה אשר בודקים, המלה SHALOM. לפיכך OT_ACHRONA שווה ל-'S' כי היא האות האחרונה (מבחינת מיקומה באלפבית הלטיני) שכלולה במלה SHALOM. ההצבה של האות S כ-OT_ACHRONA חוסכת מן התכנית את הבדיקה

של כל השלישיות שאחת או יותר מאותיותיהן ממוקמת באלפבית הלטיני אחרי האות S. לו היינו בוחרים לפרק מלה שהאות הראשונה שלה מבחינת מיקומה באלפבית הלטיני היתה E למשל והאחרונה היתה Y, היינו צריכים לשנות את הערכים של הקבועים OT_RISHONA ו-OT_ACHRONA בהתאם.

(6) בזמן הדפסה, רוצים שתופענה 15 מלים בכל שורה. כל מלה מנצלת ארבעה מקומות: שלושה לאותיות השלישיה עצמה ואחד לרווח המפריד. המשתנה MILIM_BESHURA קובע את הגבול העליון למספר המלים שתודפסנה בכל שורה.

(22)-(16) בשורות אלו יוצרים את כל השלישיות באלפבית הלטיני שמקיימות את שני התנאים האלה:

א. האותיות שונות אחת מן השנייה.

ב. המיקום של אותיות האלפבית המרכיבות את השלישיות הוא בין OT_RISHONA לבין OT_ACHRONA.

כבר הזכרנו שהערך של OT_RISHONA ו-OT_ACHRONA משתנה בהתאם לאותיות המרכיבות את המלה הנבדקת. אם מלה זו מכילה אות מסוימת יותר מפעם אחת, התכנית בכל זאת תרכיב שלישיות שמכילות אותה רק פעם אחת.

(23) בודקים אם אותיות השלישיה מהוות קבוצה חלקית של המלה המקורית שמפרקים.

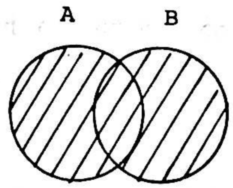
(32)-(26) בדיקה של מספר המלים בשורה. אם המספר שווה למספר המלים שנקבע מראש, יש להדפיס את השורה.

14.4 אלגברה של קבוצות

אפשר לבצע שלוש פעולות אלגבריות שונות על זוגות של קבוצות:

א. איחוד של קבוצות (Set Union)

אופרטור האיחוד מסומן A U B בתורת הקבוצות. בשפת פסקל סימנו הוא +. תוצאת האיחוד של שתי קבוצות A ו-B היא הקבוצה שמכילה את כל האיברים אשר נמצאים ב-A, ב-B או בשתיהן. נציג זאת בתיאור סכימטי, שנקרא דיאגרמת ון (Venn Diagram):



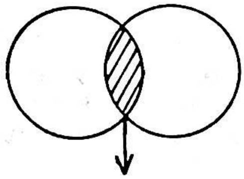
AUB

אפשר לראות מן השרטוט שקיים: A U B = B U A.
הביטוי A U B=A הוא נכון אם B היא קבוצה חלקית של A.

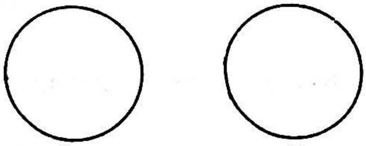
התנאי "X IN (A+B)" בודק אם האיבר X שייך לאיחוד של הקבוצות A ו-B, כלומר אם הוא נמצא ב-A, ב-B, או בשתיהן. שייכות זו נבדקת גם ע"י הביטוי הבוליאני: (X IN A) OR (X IN B).

ב. חיתוך של קבוצות (Set Intersection)

אופרטור החיתוך מסומן ע"י A ∩ B בתורת הקבוצות. בשפת פסקל סימנו הוא *. תוצאת החיתוך של שתי קבוצות A ו-B היא קבוצה שמכילה רק את האלמנטים המשותפים ב-A וב-B. כלומר, אלמנט שייך לחיתוך A∩B אם הוא שייך גם ל-A וגם ל-B. תיאור סכימטי של חיתוך מוצג בשרטוט הבא.



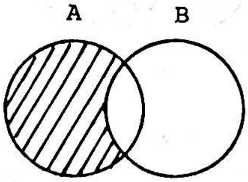
אפשר ללמוד מכך שקיים השיוויון A∩B = B∩A. כך גם קיים: A∩B = B, אם B היא קבוצה חלקית ל-A. אם φ מסמן קבוצה ריקה, נקבל A∩B=φ רק אם A ו-B הן קבוצות זרות. קבוצות זרות (Mutually Exclusive Sets) הן קבוצות, אשר אין להן אף לא איבר משותף אחד. התיאור של קבוצות זרות מוצג בשרטוט הבא:



התנאי "X IN (A*B)" בודק אם האיבר X נמצא גם בקבוצה A וגם ב-B. בדיקה שקולה נעשית ע"י התנאי: (X IN A) AND (X IN B).

ג. הפרש בין קבוצות (Set Difference)

אופרטור ההפרש בתורת הקבוצות ובפסקל הוא הסימן - (מינוס).
ההפרש (או המשלים היחסי) בין קבוצות A ו-B הוא קבוצה שמכילה כל איבר ב-A שאיננו איבר ב-B. התיאור הסכמטי של ההפרש של קבוצות A ו-B מוצג בשרטוט.



אפשר ללמוד מן השרטוט את הדברים הבאים:

- A-B איננו שווה ל: B-A, אלא אם קיים A=B ואז מקבלים: A-B=B-A=φ.
- A-B=A אם B=φ, או אם A∩B=φ. בניסוח אחר: A-B שווה ל-A אם A ו-B הן קבוצות זרות.
- A-B=φ אם A⊂B. הביטוי A⊂B כולל גם את המקרה ש: A=B.
- הביטוי הבוליאני: X IN (A-B) שקול לביטוי: (X IN A) AND NOT (X IN B)

14.4.1 יישומי אלגברה של קבוצות

כדי להסביר את נושא האלגברה של קבוצות נשתמש בדוגמא על פי ההגדרה של הקבוצות K1 עד K6 שניתנה בסעיף קודם. אלו הן קבוצות המספרים מ-1 עד 20 אשר מתחלקים ב-1, 2, 3, 4, 5 ו-6 בהתאמה.

א. K2UK3 היא קבוצת כל השלמים מ-1 עד 20 המתחלקים ב-2, ב-3 או בשניהם יחד. לכן:

```
K2+K3 = [2,3,4,6,8,9,10,12,14,15,16,18,20]
```

ב. K2+K6=K2, מכיוון שקיים K2=>K6. לכן:

```
K2+K6 = [2,4,6,8,10,12,14,16,18,20]
```

ג. אפשר לבנות קבוצה K2 בעזרת פעולת איחוד:

```
K2 = [ ]                    (* קבוצה ריקה *)
FOR I=1 TO 20 DO
   IF NOT ODD(I) THEN
      K2:= K2+[I];
```

ד. K2∩K3 היא קבוצת כל המספרים השלמים בין 1 ל-20 המתחלקים גם ב-2 וגם ב-3. קבוצה זו שקולה לקבוצת K6, כי כל מספר שמתחלק גם ב-2 וגם ב-3 מתחלק גם ב-6. כלומר, K2*K3=K6. לכן ניתן לבנות קבוצת K6 בעזרת פקודת ההשמה K6:=K2*K3.

ה. K3∩K6 היא הקבוצה של מספרים שלמים בין 1 ל-20 שמתחלקים גם ב-3 וגם ב-6. קבוצה זו שקולה ל-K6, ז"א: K3*K6=K6, מכיוון ש: K6⊂K3.

ו. K4∩K6 היא הקבוצה של השלמים בין 1 ל-20 שמתחלקים גם ב-4 וגם ב-6. ומכאן: K4*K6=[12].

ז. K5∩K6 היא הקבוצה של השלמים שבין 1 ל-20 שמתחלקים גם ב-5 וגם ב-6. זוהי קבוצה ריקה, כלומר K5∩K6=φ, כי השלם הקטן ביותר שמתחלק גם ב-5 וגם ב-6 הוא 30. נרשום זאת כך: K5*K6=[ ].

ח. K1-K2 היא הקבוצה של השלמים בין 1 ל-20 שלא מתחלקים ב-2, ז"א המספרים האי-זוגיים. לכן,

```
K1-K2= [1,3,5,7,9,11,13,15,17,19]
```

ניתן לבנות קבוצה זו כך:

```
K2:=K1;
FOR I:=1  TO  20  DO
  IF  NOT  ODD(I)  THEN
    K2:=K2-[I]
```

ט. ‏[]=K2-K1, מכיון שנתון K2⊂K1.

י. הקבוצה K6-K5 מכילה את קבוצת השלמים בין 1 ל-20 שמתחלקת ב-6 אבל לא ב-5. מכיון שקיים K6∩K5=φ, נקבל K6-K5=K6.

נניח שבתכנית PIRUK_MILA, אשר עסקנו בה בסעיף הקודם, רוצים ליצור שלישיות שמכילות לפחות תנועה (Vowel) אחת. לשם כך צריך להצהיר על משתנה קבוצתי:

```
TNUOT: SET OF 'A'..'Z';
```

בתכנית נכתוב פקודת השמה זו:

```
TNUOT:= ['A', 'E', 'I', 'O', 'U']
```

לפני שמדפיסים שלישיה, נרחיב כך את התנאי הבוליאני בשורה (23):

```
(MILA <= MILA_LEPIRUK) AND (MILA * TNUOT <> [] )
```

דורשים בתנאי המורחב הזה ש-MILA תהיה קבוצה חלקית של MILA_LEPIRUK וגם שתכיל לפחות תנועה אחת.

14.5 סדר הקדימויות של פעולות

סדר הקדימויות של הפעולות המתאימות לקבוצות זהה לסדר הקדימויות של הפעולות האריתמטיות המקבילות, כפי שמפורט להלן. לשם הדגשה משכנו קו מתחת לכל פעולה המתאימה גם לקבוצות:

- NOT
- AND MOD DIV / <u>*</u>
- <u>-</u> <u>+</u>
- <u>>=</u> <u><=</u> <u><></u> <u>=</u> <u>IN</u> < >

נציג שתי דוגמאות על פי טבלת הקבוצות K1 עד K6 שהוצגה בסעיף 14.3.1.

א. קבוצת כל המספרים האי-זוגיים בין 1 ל-20 המתחלקים ב-3 תכתב בדרך זו: ‎(K1-K2)*K3‏ ולא כך: K1-K2*K3. רמת הקדימות של חיתוך גבוהה מזו של הפרש ולכן הביטוי השני מוביל לקבוצה המכילה את ההפרש שבין:-
- קבוצת המספרים השלמים בתחום 1 עד 20,
- קבוצת החיתוך של המספרים השלמים שבין 1 ל-20 המתחלקים גם ב-2 וגם ב-3.

שני הביטויים שהוזכרו נותנים תוצאות שונות, כפי שמפורט להלן:

```
(K1-K2)*K3 = [3,9,15]
K1-K2*K3 = [1,2,3,4,5,7,8,9,10,11,13,14,15,16,17,19,20]
```

ב. קבוצת השלמים בין 1 לבין 20 שלא מתחלקים ב-3 וגם לא מתחלקים ב-5 מסומנת ע"י הביטוי ‎K1-(K3+K5)‏
או ע"י הביטוי ‎K1-K3-K5‏

<u>הערה:</u>
הקבוצה K1 נקראת <u>קבוצה חלקית ממש</u> (Proper Subset) ל-K2 אם K1⊂K2 כאשר K1<>K2. בפסקל אי אפשר לציין מצב זה בצורה ישירה, למשל ע"י כתיבה של K1<K2. הסיבה להגבלה זו היא בכך, שבדיקה כזאת היתה מעמיסה על המהדר בלי שום פרופורציה לרמת השימוש שלה. כאשר רוצים לציין או לבדוק מצב זה, צריך להשתמש בתנאי הבוליאני: ‎(K1<=K2)*(K1<>K2)‏.

14.6 קלט/פלט של קבוצות

בשפת פסקל אין אפשרות לבצע קלט/פלט ישיר של קבוצה, אלא צריך לקרוא או לכתוב את איברי הקבוצה בנפרד, זה אחר זה. כדי לקבל למשל, תדפיס של כל איבר בקבוצה אי אפשר לכתוב WRITE(A) כאשר A מוגדר כקבוצה.

בזמן הקלט מכניסים לקבוצה את הערכים שרוצים שישתתפו בה. בזמן הפלט בודקים את כל האיברים בסוג הבסיס ומדפיסים רק את אלה ששייכים לקבוצה. סוג הבסיס יכול להיות תת-טווח של השלמים או של סדרת התווים המוגדרים באותו מחשב. במהדרים מסוימים יש הגבלה חמורה לגודל של הקבוצה. אם אין הגבלה כזאת, סוג הבסיס יכול להכיל את כל סדרת התווים (כלומר, אפשר להגדיר אותו כ-CHAR).

אם סוג הבסיס של הקבוצה מתואר ע"י מנייה, אי אפשר לעשות קלט/פלט של איברי הקבוצה, אפילו כאשר מטפלים בכל אחד מהם לחוד (ישנם מהדרים משוכללים שמאפשרים זאת). הפתרון היחיד הוא לקרוא כקלט תווים או ספרות, לתרגם אותם לסוג החדש בתוך התכנית ולתרגם את הסוג החדש חזרה לתווים או ספרות בזמן הפלט. ניתן לעשות זאת ע"י השימוש בפונקציה ORD או באמצעות פקודת CASE.

עתה נציג שתי שלדות של תכניות המאפיינות סריקה וחיפוש בקבוצות. נתייחס למקרה הטיפוסי, כאשר סוג הבסיס הוא תת-טווח של INTEGER או CHAR. בקטע ההצהרות של התכנית נכתוב:

```
CONST
      GVUL_TACHTON = ...;
      GVUL_ELYON = ...;
TYPE
      TAT_TVACH = GVUL_TACHTON..GVUL_ELYON;
      SUG_KEVUTZATI = SET  OF  TAT_TVACH;
VAR
      KEVUTZA: SUG_KEVUTZATI;
```

סריקה:

סריקה נעשית כאשר רוצים להפעיל שגרה חוזרת על כל איבר בקבוצה. לשם כך נשתמש בפרוצדורה כזו:

```
PROCEDURE TIPUL_KEVUTZA (K: SUG_KEVUTZATI);
VAR
      I: TAT_TVACH;
BEGIN
      FOR  I:= GVUL_TACHTON  TO  GVUL_ELYON  DO
         IF  I  IN  K  THEN
           PROCESS (I)
END;
```

הפרוצדורה TIPUL_KEVUTZA סורקת את כל איברי הקבוצה וקובעת לגבי כל אלמנט I בקבוצה אם להפעיל עליו את PROCESS(I) או לא. PROCESS(I) יכול להיות כל תהליך שרוצים להפעיל על האלמנטים של קבוצה K:

- אם התהליך הוא הדפסה של איברי הקבוצה נוכל להחליף את PROCESS(I) בפקודה WRITE(I:2).

- כדי לקבוע את מספר האיברים של הקבוצה, יש להחליף את PROCESS(I) בפקודה: GODEL:=GODEL+1, כאשר הוא יהיה פרמטר משתנה של הפרוצדורה TIPUL_KEVUTZA. אפשרות אחרת היא להפוך את הפרוצדורה לפונקציה שלמה שמקבלת את המשתנה הקבוצתי K ומחזירה את גודל הקבוצה.

אם רוצים לעבד (או לספור) את כל האיברים בקבוצה שמקיימים תנאי מסוים, אזי PROCESS(I) יבדוק אם התנאי קיים. אם התשובה חיובית, הוא יקדם מונה או ידפיס את אותו אלמנט, בהתאם להוראת המתכנת.

<u>חיפוש:</u>

בפונקציה זו משתמשים לשם חיפוש של אלמנט מסוים אחד. מפסיקים את הביצוע כאשר מאתרים אותו ולא ממשיכים לסרוק את שאר איברי הקבוצה.

```
 (1)  FUNCTION  CHIPUS (K: SUG_KVUTZATI): TAT_TVACH;
 (2)  VAR
 (3)          I: TAT_TVACH;
 (4)          NIMTZA, NIGMAR: BOOLEAN;
 (5)  BEGIN
 (6)          NIMTZA:= FALSE;
 (7)          NIGMAR:= FALSE;
 (8)          I:= GVUL_TACHTON;
 (9)          WHILE NOT NIGMAR AND NOT NIMTZA DO
(10)               IF  (I IN K)  AND  TNAI(I)  THEN
(11)                 BEGIN
(12)                    CHIPUS:= I;
(13)                    NIMTZA:= TRUE
(14)                 END
(15)               ELSE
(16)                IF   I < GVUL_ELYON  THEN
(17)                    I:= SUCC(I)
(18)                ELSE
(19)                    NIGMAR:= TRUE;
(20)          IF  NOT  NIMTZA  THEN
(21)               WRITELN  ('אין איבר כזה')
(22)  END;                              (* סוף הפונקציה *)
```

<u>הסבר:</u>

(9) כאשר סרקנו את כל הקבוצה השתמשנו בביטוי:

FOR I:= GVUL_TACHTON TO GVUL_ELYON

כאן משתמשים בפקודת WHILE..DO אשר מאפשרת יציאה מוקדמת כאשר NIMTZA מקבל את הערך TRUE.

(10) הביטוי TNAI(I) יוחלף ע"י הדרישה הקובעת איזה איבר רוצים לאתר. אם מחפשים את הערך המספרי הקטן ביותר בקבוצה, TNAI(I) אינו נחוץ ומספיק לכתוב IF (I IN K). הלולאה תסתיים כאשר נמצא בתת-הטווח את האיבר הראשון ששייך ל-K. אם מחפשים את הערך המספרי הגדול ביותר, מתחילים את הלולאה כשמציבים I:=GVUL_ELYON בשורה (8) ובמקום שורה (17) כותבים I:=PRED(I).

בשני המקרים, של סריקת קבוצה וחיפוש של אלמנט בתוך קבוצה, רצוי להוסיף את הפקודה שמופיעה למטה כדי להמנע מעיבוד מיותר. אם K היא קבוצה ריקה, לא נבצע את הלולאה ונדפיס הודעה על כך:

```
IF K=[] THEN
   WRITELN ('הקבוצה ריקה')
ELSE
  (* שאר פקודות הפונקציה *)
```

14.7 יישומים של קבוצות

14.7.1 הכברה של אירטוסתינס (*)

המטרה של התכנית הבאה היא ליצור מספרים ראשוניים לפי השיטה שנקראת "הכברה של אירטוסתינס" (The Sieve of Eratosthenes): N מוגדר כמספר ראשוני אם אי אפשר לכתוב אותו כמכפלה של זוג מספרים שלמים כלשהו, חוץ מן הזוג (1, N).

כדי למצוא מספרים ראשוניים לפי שיטה זו משתמשים בשתי קבוצות. הקבוצה MUAMADIM מכילה את כל המספרים השלמים מ-2 עד 64 (תת טווח של כל המספרים השלמים) שמתוכם נבחר את המספרים הראשוניים. במצב ההתחלתי כל המספרים בתת-טווח שהוזכר שייכים לקבוצה זו. הקבוצה RISHONIM תכיל את כל המספרים הראשוניים שייבחרו. במצב ההתחלתי, לפני שיש תוצאות כלשהן, קבוצה זו חייבת להיות ריקה.

במהלך התכנית סורקים את המספרים בתוך MUAMADIM ומעבירים את המספרים הראשוניים לקבוצת RISHONIM. את האחרים, שאינם מספרים ראשוניים, מוחקים מקבוצת MUAMADIM. בסוף התכנית, קבוצת RISHONIM תכיל את כל המספרים הראשוניים שבין 2 לבין 64. קבוצת MUAMADIM תהיה קבוצה ריקה, שהרי את המספרים הראשוניים העברנו ל-RISHONIM ומחקנו את אלה שאינם ראשוניים.

התכנית, אשר פועלת בשיטה זו, כתובה להלן:

```
PROGRAM MISPARIM_RISHONIM (OUTPUT);
CONST
    TACHTON = 2;
    ELYON = 64;
TYPE
    KEVUTAT_SHLEMIM= SET OF TACHTON..ELYON;
VAR
    MUAMADIM, RISHONIM: KEVUTZAT_SHLEMIM;
    I, J: INTEGER;
PROCEDURE  KTOV_KEVUTZA (K: KEVUTZAT_SHLEMIM);
  VAR
        RISHON: BOOLEAN;
        I: TACHTON..ELYON;
  BEGIN                    (* KTOV KEVUTZA *)
    WRITE ('[');
    RISHON:= TRUE;
```

```
(17)        FOR  I:= TACHTON  TO  ELYON  DO
(18)           IF  I  IN  K  THEN
(19)             BEGIN                                (* כאשר המספר איננו *)
(20)               IF  NOT  RISHON  THEN   (* הראשון, כותבים פסיק *)
(21)                   WRITE (',')          (* לפני שמדפיסים אותו. *)
(22)               ELSE
(23)                   RISHON:= FALSE;  (* במספר הראשון לא כותבים *)
(24)               WRITE (I:2)             (* פסיק והופכים את הערך *)
(25)             END;                         (* של RISHON ל-FALSE *)
(26)        WRITELN(']')
(27)      END;                     (* KTOV_KEVUTZA *)
(28)    BEGIN                    (* תכנית ראשית *)
(29)      MUAMADIM:= [TACHTON..ELYON];
(30)      RISHONIM:= [ ];
(31)      I:= TACHTON;
(32)      WHILE  MUAMADIM <> [ ] DO  (* מצא את המספר הראשוני הבא *)
(33)          BEGIN
(34)            WHILE NOT (I IN MUAMADIM) DO
(35)              I:=I+1;          (* מצא את הערך הקטן ב-MUAMADIM *)
(36)            RISHONIM:= RISHONIM + [I];
(37)            J:= I;
(38)            WHILE J <= ELYON  DO (* מחק אותו ואת הכפולות שלו *)
(39)              BEGIN
(40)                MUAMADIM:= MUAMADIM - [J];
(41)                J:= J+I
(42)              END
(43)          END;
(44)      WRITE ('RISHONIM = ');
(45)      KTOV_KEVUTZA (RISHONIM)
(46)    END.
```

הפלט של התכנית:

RISHONIM = [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61]

<u>הסבר:</u>

(32) במהלך התכנית מוציאים בזה אחר זה את כל המספרים מקבוצת המועמדים. אם הם מספרים ראשוניים הם מועברים לקבוצה RISHONIM, אחרת הם נמחקים. התהליך נפסק כאשר אין יותר מספרים לבדיקה, דהיינו כאשר MUAMADIM תהיה קבוצה ריקה.

(35)-(34) השיטה למציאת המספרים הראשוניים מבוססת על העובדה שהמספר הראשון שנמצא ב-MUAMADIM הוא תמיד מספר ראשוני. במשך התהליך מוחקים את כל הכפולות של אותו מספר ראשוני וכתוצאה מכך נוצרים מקומות ריקים ברשימת המספרים בקבוצה MUAMADIM. המספרים שמלאו מקומות אלה לפני שנמחקו מקבוצת המועמדים, היו מספרים שאינם ראשוניים ולכן אין כל ענין לטפל בהם.

בפקודות (34)-(35) בלולאה זו מוסיפים 1 למשתנה I עד שערכו שווה למספר הקטן ביותר שנשאר עדיין בקבוצת המועמדים. אותו מספר חייב להיות מספר

ראשוני. בפעם הראשונה, כאשר I=2, פקודות אלו לא תתבצענה כי התנאי NOT(I IN MUAMADIM) אינו אמת. גם לאחר שמגדילים את I ל-3 התנאי לא יתקיים ולא נבצע פקודות אלו, אבל כאשר I מגיע ל-4 המצב שונה. כעת I כבר נמחק מקבוצת המועמדים כי הוא כפולה של 2 (פקודות (38)-(42)) ולכן התנאי NOT(I IN MUAMADIM) כן יתקיים. נבצע פקודה (35) ונגדיל את I ל-5. עכשיו התנאי שוב אינו מתקיים ונמשיך בפקודה (36) כמו קודם.

(36) העברה של המספר הקטן בקבוצת המועמדים לקבוצת המספרים הראשוניים.

(37)-(42) המשתנה J מסמל את המספר שנמחק מקבוצת המועמדים. מוחקים מספרים מקבוצה זו בגלל שתי סיבות:

א. המספר הנבדק הוא ראשוני. המספר כבר הועבר לקבוצה RISHONIM.

ב. המספר הוא כפולה של מספר ראשוני ועל כן אינו ראשוני.

השוואה של J ל-I בשורה (37) מדגימה את הסיבה הראשונה, שהרי I הוא ראשוני והשוואתו ל-J מאפשרת הורדתו מקבוצת המועמדים. קידום J ל-J+I בשורה (41) מדגים את הסיבה השנייה, שהרי בהתחלה J שווה ל-I. אם I ראשוני, J+I אינו ראשוני. ה-J החדש שווה ל-I*2 וכאשר מחשבים שוב J+I, הערך החדש של J יהיה I*3, שגם הוא אינו ראשוני. התהליך ממשיך עד שמוחקים את כל הכפולות של I שאינן גדולות מ-ELYON (שורה (38)), שהוא גבול התחום הנבדק. פעולת המחיקה של J נעשית בשורה (40) ע"י השוואת הקבוצה MUAMADIM להפרש של קבוצה זו והקבוצה שמכילה את האלמנט J לבדו.

(45) מדפיסים את הקבוצה RISHONIM. הפרוצדורה כתובה בשורות (10)-(27). המבנה שלה דומה למבנה של רוטינה שהוצגה קודם לעיבוד של כל איבר בקבוצה, כאשר PROCESS(I) מוחלף ע"י שורות (19)-(25).

14.7.2 ניתוח סקר

יש לכתוב תכנית כדי לרכז תוצאות סקר שנערך ולבדוק את היחס בין ההשתייכות המפלגתית של המשתתפים בסקר לבין מינם. סימולי המפלגות הם: מערך-M, ליכוד-L, אחר-A. לכל משתתף נקבע מספר זיהוי בין 1 ל-64.

התכנית רשומה להלן:

```
PROGRAM  SEKER (INPUT, OUTPUT);
CONST
        TACHTON = 1;
        ELYON = 64;
TYPE
        KEVUTZAT_ZEHUT= SET OF TACHTON..ELYON;
VAR
        GEVARIM_MARACH, GEVARIM_LIKUD,
        NASHIM_MARACH, NASHIM_LIKUD,
        GEVARIM_SHONIM, NASHIM_SHONIM: KEVUTZAT_ZEHUT;
        ZEHUT: TACHTON..ELYON;
        MIN, MIFLAGA: CHAR;
```

```
FUNCTION GODEL_KEVUTZA (K: KEVUTZAT_ZEHUT): INTEGER;
     VAR
       I, SCHUM: TACHTON..ELYON;
    BEGIN
       SCHUM:= 0;
       FOR  I:= TACHTON  TO  ELYON  DO
          IF  I  IN  K  THEN
            SCHUM:= SCHUM+1;
       GODEL:= SCHUM
    END;

PROCEDURE KTOV_KEVUTZA (K: KVUTZAT_ZEHUT);
    VAR
        RISHON: BOOLEAN;
        I: TACHTON..ELYON;
     BEGIN
        WRITE ('[');
        RISHON:= TRUE;
        FOR I:= TACHTON  TO  ELYON  DO
          IF  I  IN  K  THEN
            BEGIN
              IF  NOT  RISHON  THEN
                WRITE (',')
              ELSE
                 RISHON:= FALSE;
              WRITE (I:2)
            END;
         WRITE (']')
     END;                (* KTOV_KEVUTZA *)

BEGIN
   GEVARIM_MARACH:=[ ];
   GEVARIM_LIKUD:=[ ];
   GEVARIM_SHONIM:=[ ];
   NASHIM_MARACH:=[ ];
   NASHIM_LIKUD:=[ ];
   NASHIM_SHONIM:=[ ];
   WRITELN  ('הקש את הנתונים ללא רווח ביניהם.^^^^^^^');
   WRITELN  ('הקש תחילה מספר זהות של המשתתף.^^^^^^^^');
   WRITELN  ('אח"כ "מין":Z-זכר, N-נקבה.^^^^^^^^^^^^^^');
   WRITELN  ('הקש מפלגה מועדפת:M-מערך,L-ליכוד,A-אחרת');
   REPEAT
      READLN (ZEHUT, MIN, MIFLAGA);
      CASE  MIN  OF         (* הוסף כל אחד לקבוצה הנכונה *)
        'Z':CASE MIFLAGA OF
              'N': GEVARIM_MARACH:= GEVARIM_MARACH+[ZEHUT];
              'L': GEVARIM_LIKUD:= GEVARIM_LIKUD+[ZEHUT];
              'A': GEVARIM_SHONIM:= GEVARIM_SHONIM+[ZEHUT]
            END;
```

```
(60)            'N':CASE  MIFLAGA  OF
(61)                 'M': NASHIM_MARACH:= NASHIM_MARACH+[ZEHUT];
(62)                 'L': NASHIM_LIKUD:= NASHIM_LIKUD+[ZEHUT];
(63)                 'A': NASHIM_SHONIM:= NASHIM_SHONIM+[ZEHUT]
(64)               END
(65)          END
(66)       UNTIL EOF;
(67)                                                  (* הצג פלט *)
(68)       KTOV_KEVUTZA(GEVARIM_MARACH);
(69)       WRITELN  ('^:גברים בעד מערך');
(70)       KTOV_KEVUTZA(NASHIM_MARACH);
(71)       WRITELN  ('^:נשים בעד מערך');
(72)       KTOV_KEVUTZA(GEVARIM_LIKUD);
(73)       WRITELN  ('^:גברים בעד ליכוד');
(74)       KTOV_KEVUTZA(NASHIM_LIKUD);
(75)       WRITELN  ('^:נשים בעד ליכוד');
(76)       KTOV_KEVUTZA(GEVARIM_SHONIM);
(77)       WRITELN  ('^:גברים למפלגה אחרת');
(78)       KTOV_KEVUTZA(NASHIM_SHONIM);
(79)       WRITELN  ('^:נשים למפלגה אחרת');
(80)       WRITE (GODEL_KEVUTZA (GEVARIM_MARACH + NASHIM_MARACH));
(81)       WRITELN  ('^^^^^^= מספר המשתתפים שהצביעו בעד המערך');
(82)       WRITE (GODEL_KEVUTZA (GEVARIM_LIKUD + NASHIM_LIKUD));
(83)       WRITELN  ('^^^^^= מספר המשתתפים שהצביעו בעד הליכוד');
(84)       WRITE (GODEL_KEVUTZA (GEVARIM_SHONIM + NASHIM_SHONIM));
(85)       WRITELN  ('^= מספר המשתתפים שהצביעו בעד מפלגה אחרת'))
(86)    END.
```

<u>הסבר:</u>

(47)-(42) בתחילה, כל הקבוצות ריקות. במהלך התכנית מוסיפים כל משתתף לקבוצה המתאימה להשתייכות המפלגתית שלו.

(65)-(54) מנתחים את הקלט של כל משתתף ומצרפים את מספר הזיהוי שלו לקבוצה המתאימה, המסווגת לפי מין ושייכות מפלגתית.

(85)-(68) הפונקציה GODEL__KEVUTZA והפרוצדורה KTOV_KEVUTZA הן מן הסוג TIPUL_KEVUTZA (שהוסבר בסעיף 14.6). הן סורקות את כל הקבוצה ומבצעות תהליך כלשהוא על כל איבר בקבוצה.

<u>הקלט</u> עבור תכנית זו יכול להיראות כך:

```
            הקש את הנתונים ללא רווח ביניהם.
            הקש תחילה מספר זהות של המשתתף.
            הקש "מין":Z-זכר, N-נקבה.
    הקש מפלגה מועדפת:M-מערך,L-ליכוד,A-אחרת
    3NA
    6NA
    7ZL
    12ZM
    14NM
    17NM
```

```
22ZA
24ZL
29NL
32ZM
35ZA
36ZA
37ZL
43NA
48ZM
51ZL
53NL
55NM
58NM
60NL
62ZL^Z
```

הפלט של תכנית זו נראה כך:

```
גברים בעד מערך: [12 ,32 ,48]
נשים בעד מערך: [14 ,17 ,55 ,58]
גברים בעד ליכוד: [7, 24, 37, 51, 62]
נשים בעד ליכוד: [29, 53, 60]
גברים למפלגה אחרת: [22, 35, 36]
נשים למפלגה אחרת: [3, 6, 43]
        מספר המשתתפים שהצביעו בעד המערך =^^^^^^^7
        מספר המשתתפים שהצביעו בעד הליכוד =^^^^^^8
        מספר המשתתפים שהצביעו בעד מפלגה אחרת = 6
```

שורות ההודעה של התכנית נבנו ע"י פקודות WRITE ו-WRITELN עוקבות. הפקודה WRITE הדפיסה את הנתונים המספריים ולא קידמה את השורה ולכן התמליל הודפס ע"י הפקודה WRITELN בהמשך השורה. למשל, הפרוצדורה KTOV_KVUTZA (שורה 68) משתמשת בפקודה WRITE ואחריה בשורה (69) ישנה הפקודה WRITELN, אשר הדפיסה באותה שורה. בשורות (80) ו-(81) יש גם כן פקודות כתיבה עוקבות, אשר מכוונות להדפיס באותה שורה.

14.8 תרגילים

14.8.1 שאלות

(1) כתוב את הביטויים האלו במינוח של קבוצות:

א. ('B'<=CH) AND ('L'>=CH) כאשר CH מוגדר כמשתנה מן הסוג CHAR.
ב. A, B, C הם משתנים שלמים שמייצגים מספר בין 10 ל-15.
ג. X הוא מספר בין 20 ל-30 שאינו ראשוני.

(2) מוגדרות הקבוצות הבאות:

```
K1= [1,3,5,7,9]
K2= [2,4,6,8,10]
K3= 1,2,3,4,5]
K4= [4]
K5= [ ]
```

מהן התוצאות של הפעולות הבאות:

```
(a)  (K1+K2) - K3
(b)  K1*K2 = K4
(c)  (K1+K5) * (K2+K4)
(d)  9 IN ((K1-K2)- K3- K4)
(f)  K3 <= (K1+K2)
```

(3) עליך להוכיח בעזרת דיאגרמות ון את חוקי די-מורגן, אשר לפיהם אם A ו-B הן קבוצות, קיימת זהות בין הביטויים הבאים:
א. NOT((X IN A) AND (X IN B)) ו- NOT(X IN A) OR NOT (X IN B)
ב. NOT((X IN A) OR (X IN B)) ו- NOT(X IN A) AND NOT (X IN B)

14.8.2 תכניות

(1) עליך לשנות את התכנית PIRUK_MILA כדי שתוכל:
- לקבל כקלט כל מלה.
- למצוא את כל השלישיות שניתן להרכיב מאותה מלה.

התנאי הוא שהשלישיות מכילות אותיות שונות.

(2) עליך לשנות את התכנית PIRUK_MILA כדי שתוכל:
- לקבל כקלט כל מלה.
- למצוא את כל השלישיות שניתן להרכיב מאותה מלה.

השלישיות יכולות לכלול אות מסוימת פעמים, אם היא הופיעה פעמים במלה לפירוק. אפשר להניח שאות אחת לא מופיעה יותר מפעמיים במלה לפירוק.

(3) נתונות ההצהרות הבאות:

```
TYPE
     SUG_BASIS = 1..50;
     SUG_KEVUTZA = SET OF SUG_BASIS;
VAR
     K1, K2, K3: SUG_KEVUTZA;
```

כתוב תכנית שתדפיס את התוכן של הקבוצות האלו:
א. קבוצת האיברים ששייכים לקבוצה K1 או לקבוצה K2, אבל לא לשתיהן.
ב. קבוצת האיברים שמופיעים רק פעם אחת ב-K1, K2 ו-K3.
ג. קבוצת האיברים ששייכים לשתי קבוצות בדיוק מתוך K1, K2, K3.
ד. המשלים של K1, כלומר קבוצת האיברים בסוג הבסיס שאינם ב-K1.
ה. קבוצת האיברים בסוג הבסיס שאינם שייכים לאף אחת מן הקבוצות K1, K2, או K3.
ו. קבוצת האיברים בסוג הבסיס שלא נמצאים בשתיים מתוך שלוש הקבוצות.

(4) כתוב תכנית שקוראת תמליל באנגלית ומפיקה כפלט את מספר האותיות שהן תנועות (A,E,I,O,U) ואת מספר האותיות האחרות.

(5) כתוב תכנית שקוראת תמליל באנגלית וסופרת כמה סדרות של שתי תנועות (למשל AA, AE וכו'), או יותר, מופיעות בקטע. בפלט צריך להציג כל סדרה של שתי תנועות או יותר ואת מספר הסדרות שהודפסו.

(6) כתוב תכנית שקוראת תמליל באנגלית ומוציאה כפלט את המלה שבה ישנו המספר הגדול ביותר של אותיות שונות. קבע את מספר האותיות השונות ע"י הגדרה של כל מלה כקבוצה ובדיקה של גודל הקבוצה.

(7) בשירות שידוכים מנהלים במחשב שני קבצים. אחד מחזיק שמות של בנים ואחד מחזיק שמות של בנות. מבנה הרשומה שווה בשני הקבצים, כפי שמפורט בטבלה.

העמודות 63 עד 70 מגדירות נושאים שונים, שעבורם האות K מציינת תשובה חיובית והאות L מציינת תשובה שלילית.

עמודה	תוכן
1 - 20	שם
21 - 50	כתובת
51 - 60	טלפון
61 - 62	גיל
63	דתי (DATI)
64	תואר ראשון (TOAR)
65	בעד שחרור האישה (SHICHRUR)
66	אוהב פוליטיקה (POLITICA)
67	אוהב מוסיקה (MUSICA)
68	רוצה ילדים (YELADIM)
69	עוסק בספורט (SPORT)
70	אוהב תאטרון (THEATER)

כתוב תכנית כדי למצוא עבור בן מסוים (או בת מסוימת) את שלוש הבנות (או שלושת הבנים) עם מספר התכונות המשותפות הרב ביותר.

רמז: יש למצוא את החיתוך בין קבוצת התכונות של הבן המסוים (או הבת) לבין קבוצת התכונות של כל הבנות (או הבנים).

פרק 15
רקורסיה

15.1 דוגמאות בסיסיות

פונקציה F נקראת <u>פונקציה רקורסיבית</u> (Recursive Function), אם היא קוראת לעצמה. כלומר, אם אחת מן הפקודות המרכיבות את הפונקציה גורמת לקריאה חוזרת שלה.

15.1.1 פונקציה רקורסיבית

לדוגמא, נבחן את הפונקציה CHEZKA(X,N) שמחזירה את הערך של X בחזקת N. פונקציה זו יכולה לכלול בתוכה לולאה (ז"א פתרון איטרטיבי) ואז היא תיראה כך:

```
(1)   FUNCTION CHEZKA (X: REAL; N: INTEGER): REAL;
(2)   VAR
(3)      I: INTEGER;
(4)      Y: REAL;
(5)   BEGIN
(6)      Y:=1;
(7)      FOR I:=1  TO  N  DO
(8)         Y:= X*Y;
(9)      CHEZKA:= Y
(10)  END;
```

<u>הסבר:</u>

הפונקציה מבצעת N פעמים את הלולאה בשורות (7)-(8).

בתחילה, Y=1 והמכפלה הראשונה X*Y היא למעשה X*1. בכל שלב הפונקציה מחשבת את המכפלה של שני המשתנים X ו-Y שערכם ידוע לה. תוצאת המכפלה מהווה את הערך החדש עבור Y, לביצוע המכפלה הבאה. לאחר המכפלה הפשוטה X*1, הפונקציה עוברת למכפלה X*X שהיא X^2, ממשיכה עם המכפלה $X*X^2$ שהיא X^3 וכן הלאה. המכפלה האחרונה היא $X*X^{N-1}$ שתוצאתה X^N ואז היא מפסיקה.

אפשר לגשת לבעיה זו בגישה שונה. במקום להתחיל במקרה הפשוט ביותר X*1, דהיינו X בחזקת אחד, נוכל להתחיל מיד במקרה היעד X בחזקת N ולהתבסס על שתי המשוואות הבאות:

א. CHEZKA (X,0) = 1

ב. CHEZKA (X,N) = X * CHEZKA (X,N-1)   עבור N > 0.

נוכל עתה לכתוב את הפונקציה הרקורסיבית הבאה:

```
FUNCTION CHEZKA (X: REAL; N: INTEGER): REAL;
BEGIN
  IF  N=0  THEN
    CHEZKA:=1
  ELSE
    CHEZKA:= X * CHEZKA(X, N-1)
END;
```

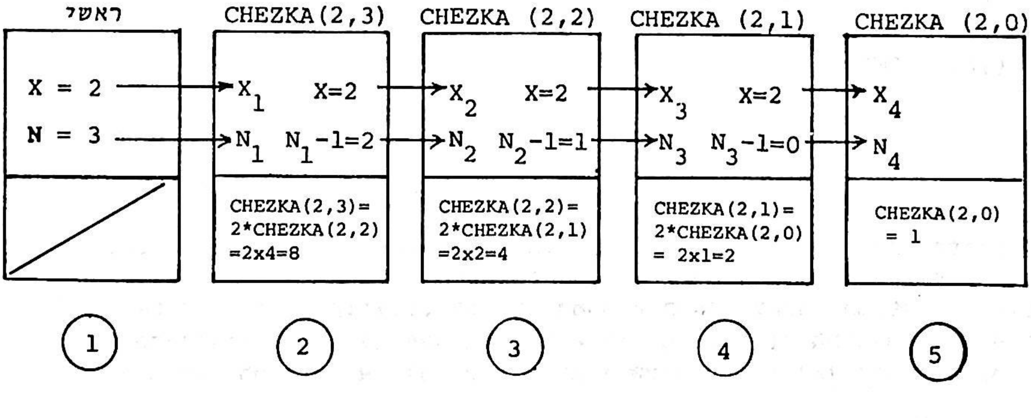
<u>הסבר:</u>

בפונקציה זו אין לולאה מפורשת, אלא רק פקודת IF כדי לבדוק אם הערך של N הוא 0 או חיובי. מתחילים לטפל מיד במקרה היעד וכותבים אותו כמכפלה של X ושל CHEZKA(X,N-1). אבל כאן קיים מצב חדש, אשר אינו מוכר: הערך של הפונקציה CHEZKA(X,N-1) לא ידוע.

עד עכשיו הדגשנו שבצד ימין של ביטוי אסור להשתמש במשתנה שערכו לא ידוע. החידוש של רקורסיה הוא, שעל אף האיסור לרשום בצד ימין של ביטוי משתנה בלי ערך, מותר לרשום בו פונקציה בלי ערך. מכיון שרקורסיה מותרת, תמיד אפשר לקרוא לפונקציה CHEZKA(X,N-1) ולקבל כתוצאה מקריאה זו את הערך שלה. בדרך זו מתקיימת הדרישה שבצד ימין של הביטוי יהיה משתנה שערכו ידוע.

הקושי אשר נובע מהטיפול בביטוי שהמשתנה בצידו הימני אינו ידוע, גרם לכך שהרקורסיה אינה מותרת בכמה שפות תכנות.

נציג בדוגמא את חישוב הערכים ע"י פונקציה רקורסיבית:
נתונה תכנית שבה נמצאת הפקודה Y=:CHEZKA(2,3). נבחן את הערך שמתקבל לאחר ביצוע הפקודה הזו בתכנית הראשית.



<u>הסבר:</u>
הקריאה ע"י התכנית הראשית מעבירה את הפרמטרים X=2, N=3 לפונקציה CHEZKA. הפונקציה CHEZKA איננה יכולה להעביר ערך כלשהוא בשלב הזה. היא מחזיקה את הפרמטרים בשמות X_1 ו-N_1, מחשבת את X_2 ו-N_2 וקוראת שוב לפונקציה CHEZKA, עד שהיא מגיעה לשורה (6) של הפונקציה, שם מבצעים:

CHEZKA:= X_2 * CHEZKA(X_2,N_2)

הפונקציה CHEZKA(2,2) מוצגת במשבצת (3) בשרטוט. דבר דומה קורה גם במשבצת (4) ורק במשבצת (5) המצב משתנה. עלינו לחשב את הערך של הפונקציה CHEZKA(2,0) והנה מגלים שהוא כתוב במפורש בשורה (4) של תכנית הפונקציה ושווה ל-1. העובדה שמגיעים למצב שהערך של הפונקציה ידוע מיד ואין צורך להמשיך ולחשב, איננה מקרה בלבד, אלא דרישת קדם כאשר כותבים תכנית רקורסיבית.

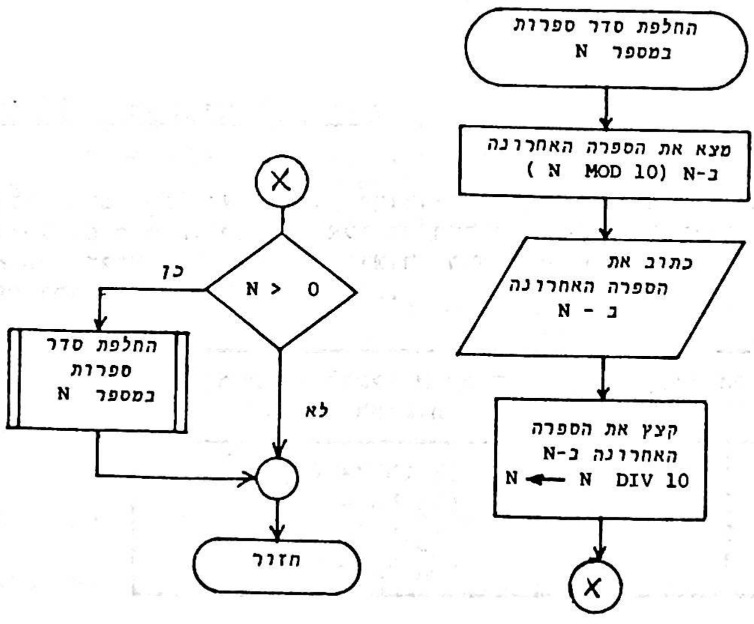
בכל חישוב, גם כאן צריך להגיע לערך סופי של הפונקציה. תוצאה זו מקבלים במשבצת (5) וצריכים לחזור איתה לנקודת המוצא, אל משבצת (4), כפי שרשום בחלק התחתון של המשבצת למטה מן הקו. בשלב זה ידועים כבר כל הערכים שמופיעים בצד ימין של המשוואה ולכן נוכל לומר שקיים:

CHEZKA(2,1) = 2 * CHEZKA(2,0) = 2

תוצאה זו מעבירים למשבצת (3) והיא מאפשרת לחשב את ערכו של הביטוי באותה משבצת. מציבים תוצאה זו במשבצת (2) ובעזרתה מגיעים לתוצאה הסופית - 8. ערך זה מחזירים לתכנית הראשית במשבצת (1).

15.1.2 פרוצדורה רקורסיבית

פרוצדורה יכולה גם כן להיות רקורסיבית כפי שנראה בתכנית להחלפה של סדר הספרות במספר נתון. נתאר את השיטה בעזרת תרשים זרימה.



התכנית בשפת פסקל:

```
(1)  PROCEDURE HACHLAFAT_N (N: MISPAR_CHIYUVI);
(2)     BEGIN
(3)        WRITE (N  MOD  10);
(4)        IF  N  DIV  10 <> 0  THEN
(5)          HACHLAFAT_N (N DIV 10)
(6)     END;
```

הן בתרשים הזרימה והן בתכנית אין הוראת לולאה, אלא רק פקודת IF. הלולאה נוצרת כתוצאה מקריאות נשנות של הפרוצדורה. מובטח שקריאות אלו תסתיימנה, שהרי חילוק חוזר של N ל-10 יביא בסוף של התהליך לתוצאה של 0 ולהפסקת הביצוע.

נעקוב אחרי ביצוע התכנית ע"י המקרה הפרטי שבו N שווה 327, כלומר רוצים לבצע: HACHLAFAT N(327). להלן רשומות הפקודות לפי סדר הביצוע בפועל:

(1)		WRITE (7)	
(2)	בצע	HACHLAFAT_N (32)	
(3)		WRITE (2)	
(4)	בצע	HACHLAFAT_N (3)	
(5)		WRITE (3)	
(6)	הפרוצדורה	HACHLAFAT_N (3)	הסתיימה, כי הגענו לשורה (6).
(7)	הפרוצדורה	HACHLAFAT_N (32)	הסתיימה, כי קראה ל-HACHLAFAT_N(3), אשר זה עתה הסתיימה.
(8)	הפרוצדורה	HACHLAFAT_N (327)	הסתיימה, כי קראה ל-HACHLAFAT_N(32) שזה עתה הסתיימה.

הפרוצדורה הרקורסיבית הסתיימה מכיון שהתנאי בפקודת IF הופך להיות FALSE בסופו של התהליך. הדבר גורם לסיום של הקריאה האחרונה ל- HACHLAFAT_N ומאפשר לפרוצדורה הקוראת להסתיים. העובדה שהפרוצדורה הקוראת הסתיימה מאפשרת לקריאה הקודמת להסתיים וכך הלאה.

15.2 משתנים לוקליים

בכל פעם שפונקציה או פרוצדורה קוראת לעצמה, היא יוצרת קבוצת משתנים לוקליים חדשים. קבוצות אלו משתחררות רק כאשר הפרוצדורה או הפונקציה שיצרה אותן מגיעה לסיומה. קבוצות המשתנים הלוקליים בדוגמאות הקודמות וסדר שחרורן מתואר בטבלאות הבאות:

משתנים לוקליים עבור דוגמת החזקות	סדר השחרור
$(X_1,N_2)= (2,3)$	4
$(X_2,N_2)= (2,2)$	3
$(X_3,N_3)= (2,1)$	2
$(X_4,N_4)= (2,0)$	1

משתנים לוקליים עבור דוגמת ההחלפה של סדר הספרות	סדר השחרור
N_1 =327	3
N_2 =32	2
N_3 =3	1

בתכנית להחלפה של סדר הספרות ניתן גם כן לעשות לולאה פנימית במקום סדרה רקורסיבית, כמו שנראה להלן:

```
PROCEDURE HACHLAFAT_N (N: MISPAR_CHIYUVI);
    BEGIN
      REPEAT
          WRITE (N MOD 10);
          N:= N  DIV  10
      UNTIL  N=0
    END;
```

נציג דוגמא לפרוצדורה, בה המשתנה הלוקלי הוא מסוג CHAR:
תכנית להחלפת סדר התווים במחרוזת בשפה האנגלית, שמסתיימת בנקודה.

```
(1)  PROCEDURE HACHLAFAT_KELET;
(2)    VAR
(3)          TAV: CHAR;
(4)    BEGIN
(5)       READ (TAV);
(6)       IF  TAV  <> '.'  THEN
(7)           HACHLAFAT_KELET;
(8)       WRITE (TAV)
(9)    END;
```

<u>הסבר</u>:

בכל פעם שקוראים תו שונה מנקודה, נוצרת קריאה רקורסיבית לפרוצדורה HACHLAFAT_KELET. כל קריאה כזו יוצרת גירסה נוספת של המשתנה הלוקלי TAV, ז"א TAV_1, TAV_2 וכו'. כאשר מגיעים לסוף המחרוזת שגודלה N וקוראים נקודה (ז"א: TAV_{N+1}='.'), מבצעים את שורה (8) בפעם הראשונה ומדפיסים את הנקודה.

הביצוע של שורה (8) מסיים את הקריאה לפרוצדורה HACHLAFAT_KELET ומאפשר לפרוצדורה הקודמת, שהופסקה בשורה (7), לעבור לשורה (8) ולהסתיים. תהליך זה נמשך עד לקריאה הראשונה של הפרוצדורה, שבה מדפיסים את התו הראשון של הקלט ומסיימים. בזה מסתיים הביצוע של כל סדרת הקריאות.

נניח שהקלט הוא "ABCD.". סדר הפעולות לפי הפרוצדורה יהיה:

א. קריאה לפרוצדורה HACHLAFAT_KELET בפעם הראשונה. היא קוראת כקלט את האות A, שתהיה TAV_1.
ב. מבלי לסיים את הפרוצדורה הנוכחית, הפרוצדורה HACHLAFAT_KELET נקראת

בפעם השנייה ועכשיו היא קוראת כקלט את האות B שתהיה TAV_2.

ג. מבלי לסיים את הקריאה הנוכחית, HACHLAFAT_KELET נקראת פעם שלישית ואז היא קוראת כקלט את האות C שתהיה TAV_3.

ד. מבלי לסיים את הקריאה הנוכחית, HACHLAFAT_KELET נקראת בפעם הרביעית ואז היא קוראת כקלט את האות D שתהיה TAV_4.

ה. מבלי לסיים את הקריאה הנוכחית, HACHLAFAT_KELET נקראת בפעם החמישית ואז היא קוראת כקלט את התו '.' שיהיה TAV_5. קריאת הנקודה מאפשרת לסיים את השורות (7)-(6), לעבור לשורה (8), להדפיס TAV_5 (הנקודה) ובזה לסיים את הקריאה החמישית.

ו. עם סיום הקריאה החמישית אפשר להמשיך בקריאה הרביעית במקום שהפסקנו לפני הפקודה להדפסת TAV_4 בשורה (8). לכן מדפיסים עכשיו את האות D ומסיימים בכך את הקריאה הרביעית.

ז. עכשיו אפשר להמשיך בקריאה השלישית מן המקום שהפסקנו לפני הפקודה להדפסת TAV_3, שערכו C. נדפיס אותו ונסיים בזה את הקריאה השלישית.

ח. עכשיו אפשר לחזור לקריאה השנייה, להדפיס TAV_2=B ולסיים את הקריאה השנייה.

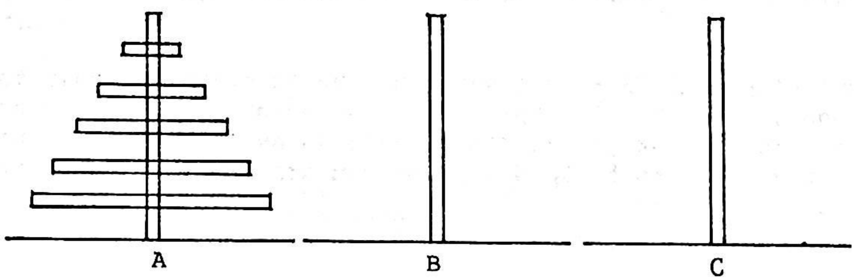
ט. עכשיו אפשר לחזור לקריאה הראשונה, להדפיס TAV_1=A ולסיים את הקריאה הראשונה ואת סדר הקריאות כולו.

אפשר להחליף את סדר הקלט גם בלי קריאה רקורסיבית של פונקציה: קולטים את התווים למערך ומציגים את תוכנו מסופו עד תחילתו. למעשה, זה מה שהקריאה הרקורסיבית עושה, אך במקום לשים את התווים במערך שמוגדר ע"י המתכנת, היא בונה מערך פנימי כתוצאה מן הקריאות הנשנות לפרוצדורה.

15.3 מגדלי הנוי - הדגמת רקורסיה (*)

טקס של כוהני הדת במגדל העתיק של ברהמה בטיבט מתואר כך:

נתונים שלושה מוטות (השמאלי יקרא A, האמצעי - B, והימני - C) ונתונות N טבעות בקטרים שונים. במצב ההתחלתי כל הטבעות נמצאות על מוט A. הטבעת N, שקוטרה הוא הגדול ביותר מונחת למטה. הטבעת שקוטרה קטן יותר ומספרה N-1 מונחת עליה וכו', עד הטבעת שקוטרה הוא הקטן ביותר שמספרה 1, שנמצאת בראש ערימת הטבעות. המצב מתואר בציור הבא:



התפקיד שהוטל על כוהני הדת היה להעביר את כל הטבעות ממוט A אל מוט C, כאשר B יכול לשמש כמוט בינים, או מוט עזר, לפי התנאים האלה:

א. מותר להעביר רק את הטבעת העליונה בכל מוט.
ב. אסור שבשלב כלשהו של התהליך טבעת גדולה תהיה מונחת על טבעת קטנה ממנה.
ג. מוטות A ו-C יכולים לשמש כמוטות ביניים, אם התנאים שהוזכרו מתקיימים.

אפשר לפתור בעיה זו בשיטה רקורסיבית. ניתן לתאר את שלבי הפתרון בעזרת 4 צעדים אלה, כאשר N הוא מספר הטבעות:

א. אם N=1, מעבירים את הטבעת היחידה מ-A ל-C ועוצרים.
ב. אם N>1, מעבירים את N-1 הטבעות העליונות ממוט A אל מוט B, כאשר C משמש כמוט ביניים.
ג. לאחר שמעבירים N-1 טבעות ל-B, מעבירים ממוט A למוט C את הטבעת האחרונה, שקוטרה גדול יותר.
ד. הטבעת שקוטרה הוא הגדול ביותר נמצאת במוט היעד C. עתה אפשר להמשיך ולהעביר למוט C את N-1 הטבעות שנמצאות במוט B, כאשר מוט A משמש כמוט ביניים.

בצעדים ב ו-ד מניחים שעל אף שלא ידוע איך להעביר N טבעות ממוט למוט, כן יודעים איך להעביר N-1 טבעות ממוט למוט. השאלה המתעוררת פה היא האם אומנם יודעים זאת? והתשובה היא, לא! אבל יודעים איך להציג העברה של N-1 טבעות בשני שלבים: בתחילה מעבירים טבעת אחת ואח"כ מעבירים (N-1)-1 טבעות. אם ממשיכים מגיעים למקרה של העברת טבעת אחת (ז"א; נפחית N-1 פעמים את המספר 1) והפתרון למקרה זה ידוע מצעד א למעלה.

התכנית הבאה מבוססת על 4 הצעדים שפורטו קודם:

```
PROGRAM MIGDAL (INPUT,OUTPUT);
TYPE
      MISPAR_SHALEM = 1..MAXINT;
VAR
      N: MISPAR_SHALEM;

PROCEDURE MIGDALEI_HANOI (N: MISPAR_SHALEM;
       MOT_MAKOR, MOT_YAAD, MOT_EZER: CHAR);
BEGIN
                         (* אם יש רק טבעת אחת, העבר אותה למוט
                         היעד וסיים את הפרוצדורה. *)
  IF  N=1  THEN
    WRITELN (MOT_YAAD, 'למוט', MOT_MAKOR, 'העבר טבעת 1 ממוט')
  ELSE
   BEGIN
      (* העבר N-1 טבעות מ-A ל-B כאשר C משמש כמוט ביניים *)
     MIGDALEI_HANOI (N-1, MOT_MAKOR, MOT_EZER, MOT_YAAD);
      (* העבר ממוט A למוט B את הטבעת שנשארה *)
     WRITELN(MOT_YAAD,'למוט',MOT_MAKOR,'ממוט',N,'העבר טבעת');
      (* העבר N-1 טבעות מ-B ל-C, כאשר A משמש כמוט ביניים *)
     MIGDALEI_HANOI (N-1, MOT_EZER, MOT_YAAD, MOT_MAKOR);
   END
END;                    (* פרוצדורה MIGDALEI_HANOI *)
```

```
(23)   BEGIN                              (* MIGDAL התכנית *)
(24)      READ(N);
(25)      MIGDALEI_HANOI (N, 'A', 'B', 'C')
(26)   END.
```

נעקוב אחר מסלול ביצוע התכנית באמצעות הצגה חלקית של הפלט למקרה N=5, ז"א כאשר נתונות 5 טבעות. מכיון שיש שתי קריאות לפרוצדורה הרקורסיבית, צריך לעקוב אחר הקריאה הרקורסיבית וגם אחר המקום בתכנית שבו נעשית קריאה זו.

מסלול הפתרון מוצג בתרשים שבעמוד הסמוך.

<u>שורות פלט</u> (שורה 12 בתכנית):

1. העבר טבעת 1 ממוט A למוט C
2. העבר טבעת 2 ממוט A למוט B
3. העבר טבעת 1 ממוט C למוט B
4. העבר טבעת 3 ממוט A למוט C
5. העבר טבעת 1 ממוט B למוט A
6. העבר טבעת 2 ממוט B למוט C
7. העבר טבעת 1 ממוט A למוט C
8. העבר טבעת 4 ממוט A למוט B

<u>הסבר:</u>

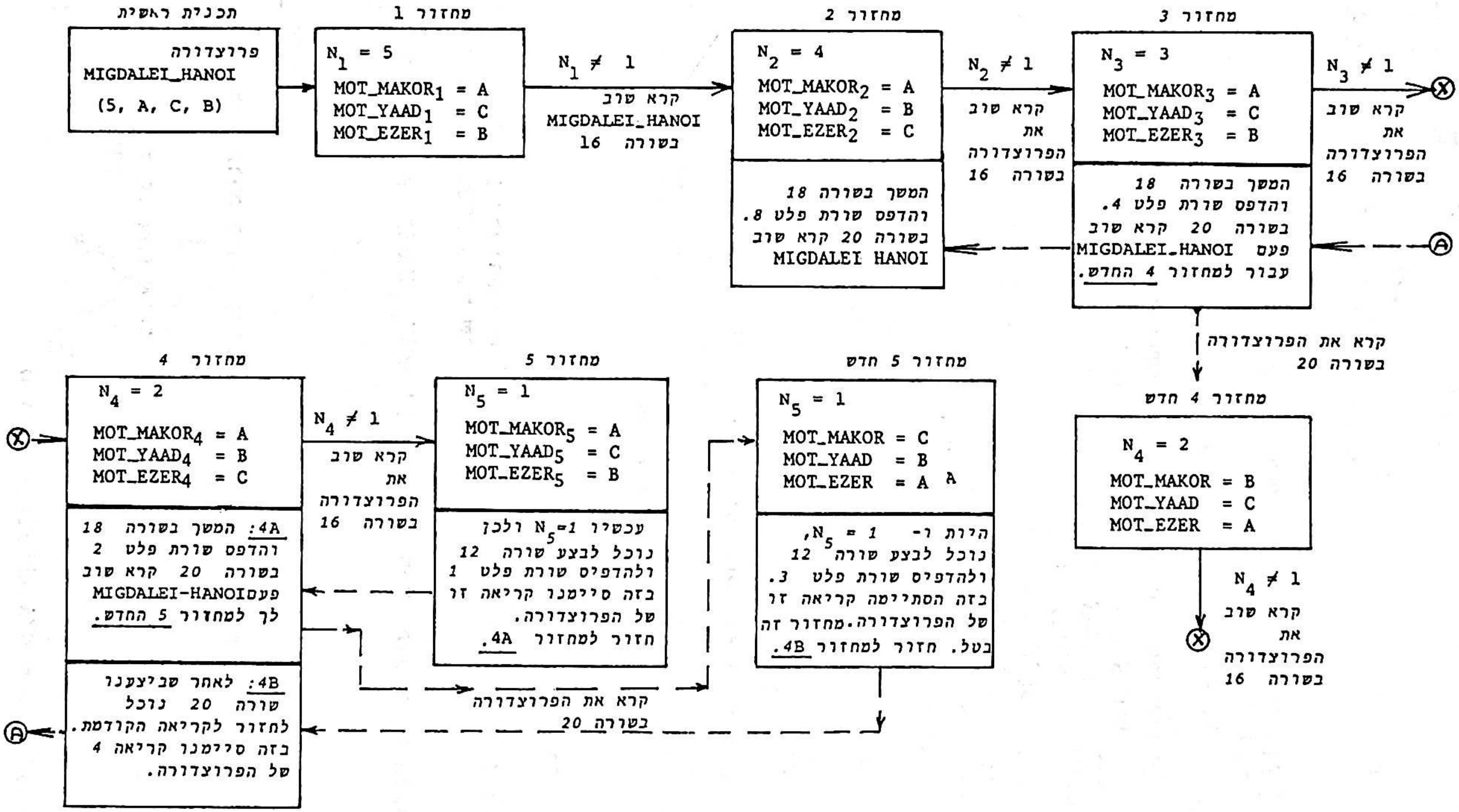
הקושי במעקב אחרי הפתרון נובע מן העובדה ששמות הפרמטרים של הפרוצדורה תמיד מתחלפים. במחזור הראשון המוט המקורי, מוט היעד ומוט הביניים הם A, C ו-B בהתאמה. במחזור 2, שמגיעים אליו בשורה (16) של הפרוצדורה, הסדר החדש הוא C, B, A (מוט הביניים B עובר למקומו של מוט היעד).

הקושי מודגש במיוחד במשבצת האמצעית של מחזור 4, שבו יש החלפה נוספת של סדר המוטות לאחר הסתעפות לביצוע חדש של הפרוצדורה משורה (20). בשורה (20) סדר הפרמטרים הוא מוט מקורי, מוט יעד ומוט ביניים דהיינו C, B, ו-A בהתאמה (עיין במחזור 5 שבתרשים המצורף).

לאחר שאוספים 4 שורות של פלט, יש במשבצת הנמוכה של מחזור 3 קריאה נוספת לפרוצדורה. הפרמטרים עכשיו הם B, C, A (עיין במחזור 4). המחשב יצטרך בשלב הזה לקרוא מחדש את הפרוצדורה פעמים נוספות. כאשר עורכים מעקב ידני אפשר לשים לב לעובדה שמחזור 4 החדש (עם N=2) שונה ממחזור 4 הישן (שכבר סיימנו) רק בזה שבמקום מוטות C,B,A סדר המוטות הוא A,C,B. לכן במקום להוציא את שורות (5), (6), (7) של הפלט ע"י יצירה מחדש של הפרוצדורה, נוכל להעתיק את שורות הפלט (1), (2), (3) אבל יש להחליף את A ב-B, B ב-C ו-C ב-A. כך קבלנו את שורות (5), (6) ו-(7) של הפלט.

בשלב זה, נוכל לחזור לשורה (20) שממנה הסתעפנו, לסיים מחזור 3 ולחזור לשורה (17) במחזור 2. בשורה (18) מדפיסים שורת פלט (8), כאשר הפרמטרים המתאימים רשומים במשבצות של מחזור 2. מכאן עוברים לשורה (20) כדי להתחיל מחזור 3 חדש (אינו מופיע בתרשים).

על פי אמונתם של כוהני הדת בטיבט, העולם יגיע לקיצו כאשר הם יסיימו בהצלחה את משחק המגדלים כשמספר הטבעות הוא 64. כאשר מנסים לעקוב אחר התהליך של העברת 5 טבעות, אפשר להבין מדוע אין כל חשש מכך שקץ העולם יגיע בקרוב.



תכנית ראשית
פרוצדורה
MIGDALEI_HANOI
(5, A, C, B)
מחזור 1
$N_1 = 5$
$MOT_MAKOR_1 = A$
$MOT_YAAD_1 = C$
$MOT_EZER_1 = B$
$N_1 \neq 1$
קרא שוב
MIGDALEI_HANOI
בשורה 16
מחזור 2
$N_2 = 4$
$MOT_MAKOR_2 = A$
$MOT_YAAD_2 = B$
$MOT_EZER_2 = C$
המשך בשורה 18
והדפס שורת פלט 8.
בשורה 20 קרא שוב
MIGDALEI HANOI
$N_2 \neq 1$
קרא שוב
את
הפרוצדורה
בשורה 16
מחזור 3
$N_3 = 3$
$MOT_MAKOR_3 = A$
$MOT_YAAD_3 = C$
$MOT_EZER_3 = B$
המשך בשורה 18
והדפס שורת פלט 4.
בשורה 20 קרא שוב
פעם MIGDALEI_HANOI
עבור למחזור 4 החדש.
$N_3 \neq 1$
קרא שוב
את
הפרוצדורה
בשורה 16
קרא את הפרוצדורה
בשורה 20
מחזור 4 חדש
$N_4 = 2$
MOT_MAKOR = B
MOT_YAAD = C
MOT_EZER = A
$N_4 \neq 1$
קרא שוב
את
הפרוצדורה
בשורה 16
מחזור 4
$N_4 = 2$
$MOT_MAKOR_4 = A$
$MOT_YAAD_4 = B$
$MOT_EZER_4 = C$
4A: המשך בשורה 18
והדפס שורת פלט 2
בשורה 20 קרא שוב
פעםMIGDALEI-HANOI
לך למחזור 5 החדש.
4B: לאחר שביצענו
שורה 20 נוכל
לחזור לקריאה הקודמת.
בזה סיימנו קריאה 4
של הפרוצדורה.
$N_4 \neq 1$
קרא שוב
את
הפרוצדורה
בשורה 16
מחזור 5
$N_5 = 1$
$MOT_MAKOR_5 = A$
$MOT_YAAD_5 = C$
$MOT_EZER_5 = B$
עכשיו $N_5=1$ ולכן
נוכל לבצע שורה 12
ולהדפיס שורת פלט 1
בזה סיימנו קריאה זו
של הפרוצדורה.
חזור למחזור 4A.
קרא את הפרוצדורה
בשורה 20
מחזור 5 חדש
$N_5 = 1$
MOT_MAKOR = C
MOT_YAAD = B
MOT_EZER = A
היות ו- $N_5 = 1$,
נוכל לבצע שורה 12
ולהדפיס שורת פלט 3.
בזה הסתיימה קריאה זו
של הפרוצדורה. מחזור זה
בטל. חזור למחזור 4B.

15.4 תרגילים

15.4.1 שאלות

(1) מה מבצעת התכנית הזו:

```
PROGRAM PYRAMID (INPUT, OUTPUT);
VAR
   I, K, SIZE: INTEGER;

PROCEDURE  HAFOCH (J, N: INTEGER);
   BEGIN
     IF  J=N  THEN
       WRITE  (N)
     ELSE
       BEGIN
         WRITE  (J);
         HAFOCH (J+1, N);
         WRITE  (J)
       END
   END;

BEGIN         (* PYRAMID *)
  READ  (SIZE);
  FOR  K:=1  TO  SIZE  DO
    BEGIN
      I:= 1;
      HAFOCH (I, K);
      WRITELN
    END
END.
```

(2) נתונה הפרוצדורה הרקורסיבית "מגדלי הנוי", שהוצגה בסעיף 15.2. סיים את תהליך המעקב, משלב 9 עד שלב 31, כאשר N=5.

(3) שנה את התכנית HACHLAFAT__KELET כדי שלא תדפיס את הנקודה בסוף המחרוזת.

15.4.2 תכניות

(1) כתוב תכנית רקורסיבית שקוראת תווים ומדפיסה אותם עד שהיא נתקלת בתו רווח.

(2) כתוב שתי תכניות, אחת איטרטיבית ואחת רקורסיבית, כדי לחשב כל אחת מן הפונקציות דלהלן:

א. N! (עצרת). נתון:

```
ATZERET(0) = 1
ATZERET(N) = N*ATZERET(N-1), N>0
```

ב. מכפלה של M ו-N. נתון: KEFEL(N,1) = M
KEFEL(N,M) = M + KEFEL(M,N-1), N>1

ג. מספר פיבונצ'י N. נתון: FIB(0) = 0, FIB(1) = 1
FIB(N) = FIB(N-1) + FIB(N-2), N>1

(3) כתוב פונקציה רקורסיבית כדי לבנות מספר שלם מסדרה של תווי ספרות. השיטה מבוססת על שימוש חוזר בנוסחה:

MISPAR:= 10 * MISPAR + ORD(C) - ORD('0')

C הוא התו שעכשיו נקרא. סדרת הספרות לבניית המספר מסתיימת כאשר פוגשים בתו שאיננו ספרה.

(4) כתוב תכנית רקורסיבית כדי להפוך מספר עשרוני למספר אוקטלי, ע"י שימוש בשיטה המתוארת, כאשר X הוא המספר העשרוני:

א. אם X=0, סיים את התהליך.
ב. אם X<>0, הספרה האוקטלית הלא משמעותית ביותר הינה: X MOD 8.
ג. החלף את X בביטוי X DIV 8.
ד. חזור לצעד א.

(5) כתוב פונקציה רקורסיבית לבצע את האלגוריתם של אוקלידס. האלגוריתם מיועד למציאת ה-GCF (GREATEST COMMON FACTOR) של שני מספרים שלמים M ו-N. זהו הערך (או "המכנה המשותף") הגדול ביותר שאפשר לחלק בו את M וגם את N. השיטה מתוארת כך:

א. אם M ו-N שווים, אפשר לבחור בכל אחד מהם כ-GCF ולסיים את התהליך.
ב. אם M ו-N שונים, יש להפחית את המספר הקטן מן הגדול, ולעבוד עם שני המשתנים האלו:
  - השלם שהיה הקטן ביותר לפני פעולת החיסור.
  - השלם החדש שהוא ההפרש בין שני המשתנים המקוריים.
ג. חזור לשלב א.



פרק 16
מבני נתונים

16.1 רשימות

16.1.1 רשימה מסודרת

רשימה מסודרת (Ordered List) היא מערך ממויין (Sorted Array), שבו כל איבר גדול מן הקודם לו. הערך של האיבר הראשון במערך זה הוא הקטן ביותר והערך של האיבר האחרון הוא הגדול ביותר. אם השם של המערך הוא NETUNIM וגודלו N, אזי נוכל לומר שקיים: NETUNIM[I+1] > NETUNIM[I] בעבור כל ערך של I בין 1 לבין N-1.

כדי להוסיף איבר לרשימה זו דרושות שלוש פעולות:

א. למצוא את המקום ברשימה, אשר מתאים בעבור האיבר החדש: למשל, בין האיבר השלישי לבין הרביעי.
ב. לפנות את המקום בעבור האיבר החדש: לשם כך יש להזיז את כל האיברים שמקומם יהיה אחרי האיבר החדש מקום אחד לאחור כלומר, איבר שמקומו היה I יעבור למקום I+1.
ג. להכניס את האיבר החדש למקום שהוכן עבורו.

נציג דוגמא: מנהל גן חיות בקיבוץ מחזיק רשימה של בעלי החיים שנמצאים ברשותו. רשימה זו מסודרת לפי האלפבית, ומופיעה במערך NETUNIM.

```
NETUNIM[1] = אווז
NETUNIM[2] = ברבור
NETUNIM[3] = חתול
NETUNIM[4] = פרה
NETUNIM[5] = שועל
```

מנהל הגן מוסיף כלב לגן החיות ומוסיף שם זה לרשימה. הצירוף של המלה "כלב" לרשימה מחייב את הצעדים הבאים:
א. מציאת המקום המתאים ברשימה עבור המלה "כלב". זהו המקום הרביעי, לפני "פרה" ואחרי "חתול".
ב. הזזת האיברים שנמצאים במקום הרביעי ואילך, ז"א "פרה" ו-"שועל".

הרשימה החדשה נראית כך:

NETUNIM[1] = אווז
NETUNIM[2] = ברבור
NETUNIM[3] = חתול
NETUNIM[4] = כלב
NETUNIM[5] = פרה
NETUNIM[6] = שועל

כדי לבטל איבר מן הרשימה סדר הפעולות שונה במקצת וכולל שני צעדים:

א. איתור האיבר שרוצים לבטל.
ב. הזזה לכיוון ראש הרשימה של כל האיברים שנמצאים אחרי האיבר שרוצים לבטל. הזזה זו היא של מקום אחד ובסופה יתפנה מקום אחד בסוף הרשימה. הביטול של האיבר המיועד למחיקה בתהליך ההזזה נעשה ע"י כתיבה של האיבר הבא אחריו ברשימה במקומו.

נובע מכך, שאם מוסיפים או מבטלים איבר בראש רשימה שאורכה N, צריכים לעשות כ-N פעולות הזזה. אם האיבר הנוסף או המבוטל נמצא בסוף הרשימה, אין צורך בפעולות הזזה כלשהן. בממוצע דרושות N/2 הזזות בכל פעולת הוספה או ביטול של איבר ברשימה.

16.1.2 רשימה משורשרת

כדי להקטין את מספר ההזזות בזמן הוספה, או ביטול של איברים של רשימה מסודרת, משתמשים בשיטה של רשימה משורשרת (Linked List), שבה שני מערכים:

א. מערך הנתונים (Data Array) מחזיק את הנתונים ללא סדר כלשהו. הנתונים יכולים להיות ערכים מספריים, רשומות, או שמות כמו בדוגמא ועוד.

ב. מערך המצביעים (Array of Pointers) מכיל את סדר האיברים של מערך הנתונים. כל איבר במערך זה מקביל לאיבר במערך הנתונים והוא מכיל מצביע אל האיבר שנמצא אחריו לפי סדר המיון של הרשימה. מצביע 0 מורה שהנתון באיבר המקביל במערך הנתונים הוא האחרון ברשימה.

למטה מוצגת רשימת גן החיות כרשימה משורשרת. המערך הראשון והשני נקראים NETUNIM ו-MATZBIA בהתאמה.

מערך הנתונים:

NETUNIM[1] = שועל
NETUNIM[2] = ברבור
NETUNIM[3] = פרה
NETUNIM[4] = אווז
NETUNIM[5] = חתול
NETUNIM[6] =

מערך המצביעים:

MATZBIA[1] = 0
MATZBIA[2] = 5
MATZBIA[3] = 1
MATZBIA[4] = 2
MATZBIA[5] = 3
MATZBIA[6] =

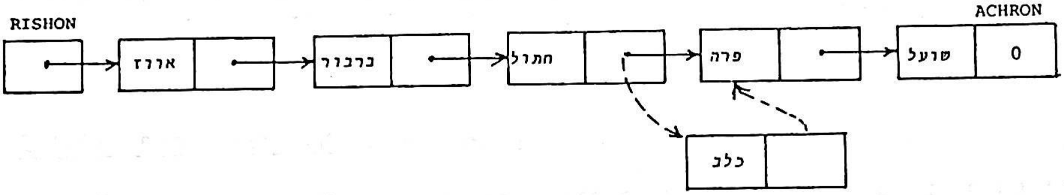
נסביר את הקשר שבין שני המערכים המרכיבים רשימה משורשרת:

א. האיבר השני במערך הנתונים מכיל את המלה "ברבור". המלה שאחריו לפי סדר המיון היא המלה "חתול", אשר נמצאת באיבר החמישי של מערך זה. בהתאם לכך באיבר השני במערך המצביעים, אשר מקביל לאיבר עם "ברבור", נמצא את המצביע 5, שפונה אל האיבר החמישי במערך הנתונים.

ב. באיבר MATZBIA[I] נמצא מצביע למקום הבא ברשימה אחרי איבר I שבמערך הנתונים. האיבר הראשון ברשימה אינו נמצא לאחר איבר אחר וכדי לדעת את מיקומו יש צורך במשתנה מיוחד, שנקרא לו RISHON. במקרה זה RISHON=4, ז"א בראש הרשימה נמצאת השם "אווז", שרשום במקום הרביעי במערך.

ג. כדי להמשיך בסריקה ברשימה בודקים את MATZBIA[4] אשר שווה ל-2. עוברים למערך הנתונים, לאיבר NETUNIM[2] אשר רשום בו "ברבור". זהו האיבר השני ברשימה מבחינת סדר האלפבית. בודקים את MATZBIA[2] אשר שווה ל-5 ועל סמך זה עוברים אל NETUNIM[5] שבו רשום "חתול", שהוא השלישי ברשימה בסדר האלפבית, על אף שהוא נמצא באיבר החמישי במערך. ממשיכים כך ל"פרה" ואחר כך ל"שועל", אשר רשום באיבר הראשון במערך הנתונים. האיבר המקביל במערך המצביעים הוא MATZBIA[1]=0, אשר מסמן את סוף הרשימה.

ייצוג סכמטי של רשימה משורשרת זו, כולל הוספת איבר, נראה כך:



כדי להוסיף איבר לרשימה צריך לבצע את הפעולות הבאות:

א. לשנות את המצביע של האיבר <u>שלפני</u> האיבר החדש כדי שיצביע עתה אל האיבר החדש.
ב. לרשום מצביע עבור האיבר החדש כדי שיצביע בצורה נכונה אל האיבר שאחריו.

בדוגמא זו, אם מוסיפים איבר שמכיל את המלה "כלב" הוא צריך להיות, לפי סדר המיון, בין המלה "חתול" לבין המלה "פרה". לפיכך צריך להחליף את המצביע של האיבר המקביל ל"חתול" כדי שיצביע אל האיבר שמכיל את המלה "כלב". את המצביע של האיבר המקביל ל"כלב" יש לכוון אל האיבר שמכיל את המלה "פרה".

באופן מפורש יותר, הוספה של המלה "כלב" לרשימה מחייבת את הוספת האיברים הבאים:

א. במערך הנתונים: NETUNIM[6]=כלב
איבר זה הוא האיבר הפנוי הראשון במערך NETUNIM. רושמים בו נתון חדש מבלי להזיז איבר כלשהוא. את סדר האיברים קובעים באמצעות המצביעים.

ב. במערך המצביעים: MATZBIA[5]:= 6
המלה "חתול" נמצאת באיבר החמישי וצריכה להיות לפני המלה "כלב" בסדר האלפביתי. המצביע של "חתול" יצביע על NETUNIM[6] שמחזיק את "כלב".

ג. במערך המצביעים: MATZBIA[6]:=3
המלה "פרה" נמצאת בסדר האלפביתי אחרי "כלב" ולכן המצביע של "כלב" יצביע על האיבר שמכיל את "פרה".

לפני ההוספה של האיבר החדש, האיבר MATZBIA[5] הכיל את הערך 3, אשר נמצא כעת באיבר MATZBIA[6]. לפני ההוספה האיבר "חתול" הצביע על האיבר "פרה", ועכשיו האיבר "כלב" מצביע עליו. זה מדגים את העובדה שאת המצביע החדש ניתן לקבל תמיד מן האיבר שמיקומו ברשימה לפני האיבר שזה עתה מוסיפים.

את האיבר החדש רושמים במקום הראשון הפנוי, כדי להמנע מהזזות כלשהן. לכאורה אפשר להניח שהמקומות הפנויים יהיו תמיד בסוף הרשימה. לשם כך אפשר להשתמש במשתנה ACHARON, אשר יצביע על המקום האחרון, כפי שמחזיקים את המשתנה RISHON להצביע על האיבר הראשון ברשימה.אולם, מכיון שמבטלים איברים בכל מקום במערך הנתונים, נוצרים מקומות פנויים גם באמצע הרשימה והבעיה של מציאת המקום הפנוי הראשון נעשית מורכבת יותר ומשתנה העזר המוצע לא יועיל.

הפתרון לבעיה זו הוא להשתמש לא רק באיברים של המערך MATZBIA שמקבילים לאיברים ה"מלאים" במערך NETUNIM, אלא גם באיברים של MATZBIA המקבילים לאיברים הפנויים שבמערך NETUNIM. נשתמש באיברים אלה כדי להצביע על האיבר הפנוי הבא. בכל פעם שמתוסף איבר פנוי, נעדכן את הערכים במערך MATZBIA כדי לשקף את המצב החדש. את <u>המציין</u> (Subscript) של האיבר הראשון הפנוי נחזיק במשתנה שנקרא PANUY.

כתוצאה, המערך MATZBIA מחזיק <u>שתי</u> רשימות משורשרות, אחת עבור האיברים המלאים ואחת עבור האיברים הפנויים.

לדוגמא, נתונה רשימה משורשרת אשר כוללת איברים "מלאים" ואיברים "פנויים". נניח שהמערכים המוצהרים הם בני 10 איברים כל אחד. נניח גם שערכי המשתנים המצביעים על האיברים הראשונים ברשימת האיברים המלאים וברשימת האיברים הפנויים הם: RISHON=6, PANUY=7 בהתאמה.

המערכים של הרשימה המשורשרת הם:

NETUNIM[1] =	שועל	MATZBIA[1] =	0
NETUNIM[2] =		MATZBIA[2] =	0
NETUNIM[3] =	ברבור	MATZBIA[3] =	8
NETUNIM[4] =		MATZBIA[4] =	2
NETUNIM[5] =	פרה	MATZBIA[5] =	1
NETUNIM[6] =	אווז	MATZBIA[6] =	3
NETUNIM[7] =		MATZBIA[7] =	9
NETUNIM[8] =	חתול	MATZBIA[8] =	10
NETUNIM[9] =		MATZBIA[9] =	4
NETUNIM[10]=	כלב	MATZBIA[10]=	5

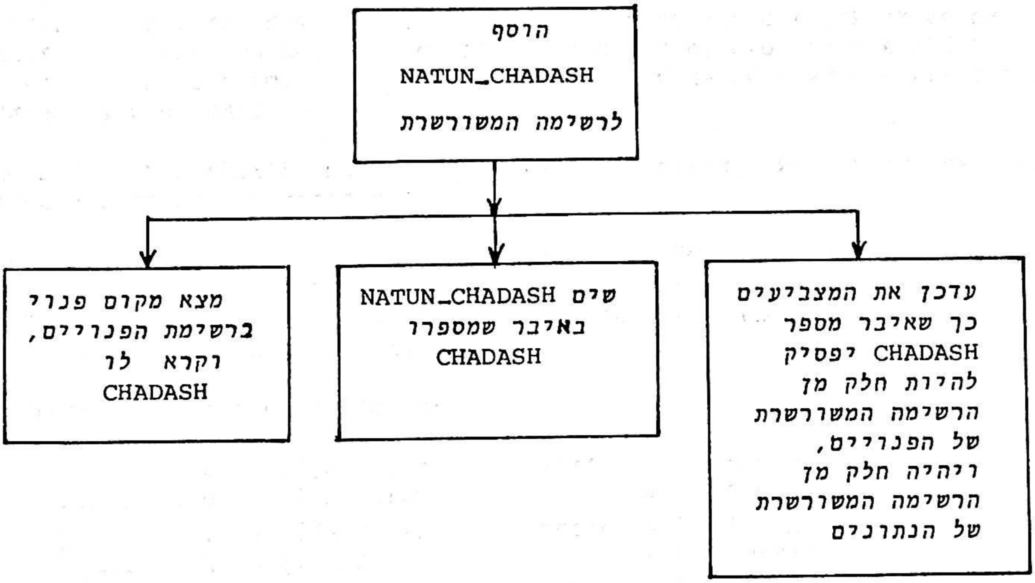
המשתנה PANUY שווה ל-7 ולכן נפנה לאיבר 7 ונמצא ש-NETUNIM[7] הוא פנוי. בתוך MATZBIA[7] נמצא הערך 9, אשר מצביע אל NETUNIM[9] וגם הוא פנוי. בדרך זו ניתן לעקוב אחרי כל רשימת האיברים הפנויים עד שמגיעים אל MATZBIA[2] שתוכנו 0. דבר זה מסמן שאיבר 2 במערך NETUNIM הוא האחרון ברשימת האיברים הפנויים.

16.2 עיבוד של רשימות

16.2.1 הוספת איברים לרשימה משורשרת

נתאר בתרשים היררכי (Hierarchy Chart) את הצעדים הבסיסיים להוספת איבר לרשימה משורשרת. בתרשים מסוג זה משרטטים מלבן נפרד עבור כל פרוצדורה, ומראים את היחס בינה לבין הפרוצדורות האחרות הנקראות על ידה או הקוראות לה. תרשים זה איננו מתאר את היחס הלוגי בין הפרוצדורות המתוארות בו.

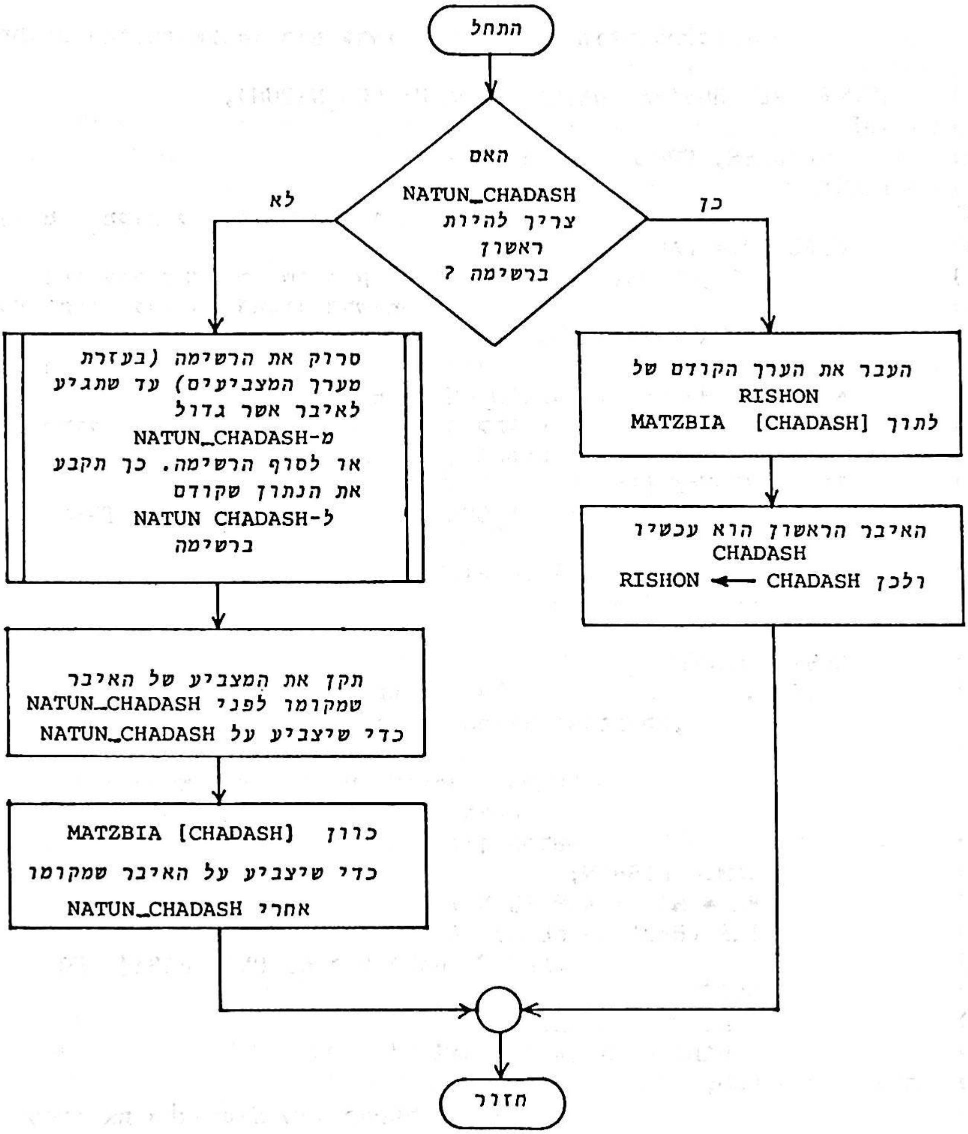
התרשים ההיררכי שמוצג כאן מתאר הוספת נתונים, אשר נמצאים במשתנה בשם NATUN_CHADASH, אל רשימה משורשרת:



תרשים הזרימה הבא מאפשר פירוט רב יותר של הפרוצדורה לעדכון המצביעים ברשימה משורשרת.

נתכנת את הפתרון על פי התרשים. רשימת המשתנים הגלובליים שיוגדרו בתכנית:

NETUNIM - מערך שמחזיק את המשתנים.
MATZBIA - מערך שמחזיק את המצביעים.
RISHON - מצביע לאיבר שמחזיק את הערך הקטן ביותר ברשימה המשורשרת.
PANUY - מצביע לאיבר הראשון ברשימה המשורשרת של תאים פנויים.
SUG_NATUN - סוג הנתונים שמופיעים במערך NETUNIM.
NULL - קבוע שמציין את סוף הרשימה וערכו שווה ל-0.


התחל
האם
NATUN_CHADASH
צריך להיות
ראשון
ברשימה ?
לא
כן
סרוק את הרשימה (בעזרת
מערך המצביעים) עד שתגיע
לאיבר אשר גדול
מ-NATUN_CHADASH
או לסוף הרשימה. כך תקבע
את הנתון שקודם
ל-NATUN CHADASH
ברשימה
העבר את הערך הקודם של
RISHON
לתוך [MATZBIA [CHADASH
האיבר הראשון הוא עכשיו
CHADASH
ולכן CHADASH ← RISHON
תקן את המצביע של האיבר
שמקומו לפני NATUN_CHADASH
כדי שיצביע על NATUN_CHADASH
כוון [MATZBIA [CHADASH
כדי שיצביע על האיבר שמקומו
אחרי NATUN_CHADASH
חזור

הערך של המשתנה RISHON יכול להיות 0 כאשר הרשימה ריקה ולכן הטווח של המציינים (Subscripts) עבור המערך NETUNIM הוא מ-0 עד N. אע"פ שלא מחזיקים נתונים באיבר [0]NETUNIM, נמצאת בתכנית ההתייחסות לאיבר NETUNIM[RISHON]. מכאן, שצריך להגדיר את [0]NETUNIM כדי שלא תהיה שגיאה בזמן ביצוע הפקודה המתייחסת לאיבר זה.

התכנית להוספה של איברים לרשימה משורשרת נתונה להלן:

```
PROCEDURE  HOSAFA (NATUN_CHADASH: SUG_NATUN);
VAR
    CHADASH, KODEM, HABA: INTEGER;
BEGIN
                                        (* מצא מקום לכתוב את הנתון החדש *)
   CHADASH:= PANUY;
                         דאג לכך שהאיבר, שהיה קודם שני ברשימת הפנויים, *)
                                          (* יהפוך להיות הראשון ברשימה זו.
   PANUY:= MATZBIA[PANUY];
                                 (* רשום NATUN_CHADASH בתא הפנוי *)
   NETUNIM[CHADASH]:= NATUN_CHADASH;
                          בדוק אם NATUN_CHADASH צריך להיות במקום *)
                                (* הראשון ברשימה המשורשרת ועדכן בהתאם.
   IF (RISHON=NULL)  OR
                        NATUN_CHADASH < NATUN[RISHON]  THEN
     BEGIN
       MATZBIA[CHADASH] := RISHON;
       RISHON:= CHADASH
     END
   ELSE
     BEGIN                          מצא את מקומו של הנתון החדש *)
                           מבחינת מיקומו הלוגי לפי סדר המיון האלפביתי,
                                              ולא מבחינת מיקומו הפיסי.
                                   הנתון החדש נמצא באיבר שמספרו CHADASH.
                                    מגיעים לכאן אם הרשימה איננה ריקה,
                                   (* והערך הקטן ברשימה קטן מן הנתון החדש.
       KODEM:= RISHON;
       HABA:= MATZBIA[RISHON];
       WHILE (HABA <> NULL)  AND
                       (NATUN_CHADASH > NETUNIM[HABA]  DO
         BEGIN
           KODEM:= HABA;
           HABA:= MATZBIA[HABA]
         END;
                                      עדכן את המצביעים כדי לאפשר הוספה. *)
       מגיעים לכאן כאשר NATUN_CHADASH <= NETUNIM[HABA], או
       כאשר NATUN_CHADASH יותר גדול מכל הערכים שהיו ברשימה
          עד עכשיו. בשני המקרים המקום עבור הנתון החדש נמצא
          לאחר המקום עם מציין שווה ל- KODEM.
          לכן, המצביע באיבר מספר KODEM צריך להצביע על
          CHADASH והמצביע באיבר מספר CHADASH צריך להצביע
          על האיבר שבא אחרי KODEM (לפני ההוספה) ומספרו
```

```
(* נמצא בתוך המשתנה ששמו HABA.
          MATZBIA[KODEM]:= CHADASH;
          MATZBIA[CHADASH]:= HABA
        END
   END;
```

בזמן בניית הרשימה, המצב ההתחלתי של RISHON הוא NULL, הערך ההתחלתי של PANUY הוא 1 וכל האיברים במערך פנויים.

הלולאה הבאה יוצרת מערך של מצביעים המתאים למצב ההתחלתי כשהרשימה ריקה ולכן היא מיוצגת על ידי מערך שכל איבריו פנויים. כל איבר פנוי מצביע אל זה שבא אחריו. את האיבר האחרון מסמנים ע"י מצביע שערכו שווה ל-0.

```
FOR I:=1 TO N-1 DO
    MATZBIA[I]:= I+1;
MATZBIA[N]:= NULL;
```

<u>16.2.2 מחיקת איברים מרשימה משורשרת</u>

הפרוצדורה הבאה מוחקת איברים מרשימה משורשרת:

```
(1)   PROCEDURE MECHIKA (NATUN_LEMECHIKA: SUG_NATUN);
(2)   VAR
(3)        KODEM, YASHAN: INTEGER;
(4)                                (* מצא את האיבר למחיקה *)
(5)   BEGIN
(6)     YASHAN:= RISHON;
(7)     WHILE NETUNIM[YASHAN] <> NATUN_LEMECHIKA  AND
(8)                        YASHAN <> NULL DO
(9)       BEGIN
(10)         KODEM:= YASHAN;
(11)         YASHAN:= MATZBIA[YASHAN]
(12)      END;
(13)                      (* נוציא את האיבר למחיקה מן הרשימה *)
(14)    IF YASHAN = NULL THEN
(15)      WRITELN ('האיבר לא נמצא ברשימה')
(16)    ELSE
(17)      IF  RISHON = YASHAN  THEN
(18)        RISHON:= MATZBIA[RISHON]
(19)      ELSE
(20)        MATZBIA[KODEM]:= MATZBIA[YASHAN];
(21)           (* הוסף לרשימת הפנויים את האיבר שהתפנה עכשיו *)
(22)    MATZBIA[YASHAN]:= PANUY;
(23)    PANUY:= YASHAN
(24)  END;
```

<u>הסבר:</u>

(12)-(6) בכל שלב מחזיקים בשני נתונים: האיבר הקודם ברשימה והאיבר הנוכחי. בשורה (6) אין איבר קודם, כי נמצאים באיבר הראשון ברשימה. אם

זהו האיבר המיועד למחיקה, לא נכנסים ללולאה שבשורות (7) עד (12). אם האיבר הראשון איננו מיועד למחיקה, נכנסים ללולאה וממשיכים. רושמים בכל שלב את האיבר הקודם ב-KODEM ואת האיבר הנוכחי ב-YASHAN עד שפוגשים באיבר למחיקה.

(14)-(15) אם YASHAN=NULL פירושו שסרקנו את הרשימה כולה ולא מצאנו את הנתון למחיקה. מדפיסים הודעת שגיאה.

(16)-(20) אם RISHON=YASHAN, לא נכנסים ללולאה של השורות (7) עד (12), כי האיבר למחיקה היה האיבר הראשון. כדי לטפל באיבר למחיקה, צריכים לשנות את הערך של המשתנה RISHON, אשר מצביע לראש הרשימה (18). אם RISHON שונה מ-YASHAN, המשמעות היא שמוחקים אחד מן האיברים האחרים. לשם כך צריך לדאוג לכך שהמצביע של האיבר שקדם לאיבר למחיקה יצביע על האיבר שבא אחריו. כלומר: MATZBIA[KODEM] := MATZBIA[YASHAN]

(22)-(23) המקום שהתפנה ברשימה כתוצאה מהמחיקה צריך להופיע מכאן ואילך ברשימת הפנויים. מכיון שמיקומו ברשימה איננו חשוב, הרי קל יהיה לשים אותו בתחילת הרשימה. כדי לעשות זאת, מוצאים את האיבר הראשון ברשימת הפנויים וגורמים לכך שהאיבר הפנוי החדש יצביע אליו, ז"א MATZBIA[YASHAN]:=PANUY. בכך קובעים שבראש רשימת הפנויים יהיה האיבר שזה עתה התפנה, ז"א PANUY:=YASHAN.

הפרוצדורה הקודמת מתאימה גם למקרה שהרשימה המשורשרת איננה מסודרת כלל. אם ידוע שהיא מסודרת, אפשר להגביל את הלולאה שבשורות(7)-(12) לבדיקה של איברים, אשר יותר קטנים מ-NATUN_LEMECHIKA. צריך אז לשנות את התוכן של שורות (13)-(20) כדי לבדוק אם:

```
NETUNIM[YASHAN] = NATUN_LEMECHIKA
```

פקודת IF החדשה תהיה:

```
IF NETUNIM[YASHAN] <> NATUN LEMECHIKA  THEN
  WRITELN ('האיבר לא נמצא ברשימה')
ELSE
 (* שורות (17)-(20) ללא שינוי *)
```

בסעיפים הקודמים, בדיון על רשימות משורשרות הצהרנו על שני מערכים: NETUNIM ו-MATZBIA. ניתן גם להצהיר על מערך אחד בלבד שאיבריו הם רשומות שבהן שני שדות: אחד עבור הנתון ואחד עבור המצביע.
נקרא לאיבר ברשימה זו בשם <u>צומת</u> (Node).

ההצהרות המתאימות לרשימה משורשרת מפורטות להלן:

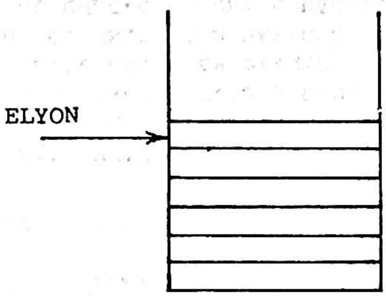
```
CONST
       MAX_RESHIMA:= 100;
TYPE
       TZOMET =
          RECORD
             NATUN: SUG_NATUN;
             MATZBIA: INTEGER
          END;
VAR
       RESHIMA: ARRAY[1..MAX_RESHIMA] OF TZOMET
```

אל הנתון בצומת I פונים כך: RESHIMA[I].NATUN
אל המצביע בצומת I פונים כך: RESHIMA[I].MATZBIA

16.3 מחסניות

ברשימה משורשרת אפשר להוסיף ולמחוק איברים בכל מקום שהוא. האפשרות הזו מקשה קצת בקידוד, אבל נותנת מרחב רב לתמרון. הטיפול ברשימת הפנויים נוח מבחינת התכנות, כי ההוספות והמחיקות הן תמיד בראש הרשימה.

מחסנית (Stack) היא רשימה משורשרת שבה כל הוספה או מחיקה מוגבלת לסוף הרשימה בלבד. שיטה זו נקראת LIFO (Last In First Out), או "נכנס אחרון יוצא ראשון". נתאר את המחסנית בשרטוט הבא:



המצביע אל האיבר האחרון נמצא במשתנה ELYON וערכו הוא המספר היחסי של האיבר ברשימה. האיבר האחרון שמוסיפים למחסנית הוא גם הראשון שמורידים ממנה, כאשר צריך להוריד איברים. הוספת איבר למחסנית מקבילה להוספת איבר בסוף רשימה משורשרת ונקראת דחיפה (Pushing). הסרת איבר ממחסנית מקבילה לביטול האיבר האחרון מרשימה משורשרת ונקראת הסרה (Popping). הוספה וביטול של איברים בסוף הרשימה הן פעולות פשוטות יחסית לפעולות הוספה וביטול במקומות אחרים ברשימה.

איברי המחסנית תמיד רצופים מכיון שלא מבטלים איברים בתוך הרשימה (מערך), אלא רק בסופה, על פי הערך שבמשתנה ELYON. לפיכך, אין צורך במערך מיוחד כדי להצביע על האיבר הבא.

בכל שלב של פעולות ברשימה יש לדעת את מספר האיברים שנמצאים במערך, ז"א את הערך הנוכחי של ELYON. הערך ההתחלתי של ELYON הוא 0, כי אז אין איברים במחסנית.

מצהירים על מחסנית בדרך זו:

```
CONST
        MAX_MACHSANIT= 100;
VAR
        MACHSANIT: ARRAY[1..MAX_MACHSANIT] OF SUG_NETUNIM;
        ELYON: 0..MAX_MACHSANIT;
```

לביצוע פעולות דחיפה והסרה דרושות שתי פרוצדורות:

```
PROCEDURE DECHIFA (NATUN: SUG_NATUN);
  BEGIN
                          (* יש עוד איבר במערך ולכן מספר
                             המקומות התפוסים עולה ב-1. *)
    ELYON:= ELYON+1;
                          (* הערך של המקום הנוסף יהיה NATUN *)
    MACHSANIT[ELYON]:= NATUN
  END;

PROCEDURE HASARA (VAR NATUN: SUG_NATUN);
  BEGIN
                          (* מעבירים את האיבר העליון למשתנה NATUN *)
    NATUN:= MACHSANIT[ELYON];
                          (* אין צורך למחוק או לאפס את האיבר העליון,
                             אלא רק לחסר 1 מן המצביע אל האיבר העליון, שהוא
                             האחרון במחסנית. כל האיברים שנמצאים לאחר
                             המצביע לאיבר העליון כאילו לא קיימים
                             ואין כל גישה אליהם, אף על פי שלא מוחקים אותם
                             ממש. כאשר מוסיפים ערכים נוספים למחסנית הם
                             יכתבו באיברים אלו והתוכן הקודם ימחק. *)
    ELYON:= ELYON-1
  END;
```

16.3.1 מצב גלישה במחסנית

עד עתה הנחנו שתמיד יהיה מקום במערך MACHSANIT כדי לדחוף איברים נוספים. גם הנחנו שלא ננסה להסיר איברים מן המחסנית כאשר פעולה זו בלתי אפשרית כי המחסנית ריקה. אולם, שתי תופעות אלו עלולות לקרות ולכן צריך לטפל במצבים אלו בהתאם. דחיפה למחסנית מלאה גורמת למצב של <u>גלישה</u> (Overflow).

הפרוצדורה המורחבת DECHIFA2 מטפלת במצב זה ע"י החזרת משתנה בוליאני שערכו TRUE כאשר מגיעים למצב של גלישה.

```
PROCEDURE DECHIFA2 (NATUN: SUG_NATUN;
                    VAR OVERFLOW: BOOLEAN);
   BEGIN
     IF ELYON = MAX_MACHSANIT THEN
       OVERFLOW := TRUE
     ELSE
       BEGIN
         OVERFLOW := FALSE;
         ELYON := ELYON + 1;
         MACHSANIT[ELYON] := NATUN
       END
   END;
```

ניסיון של הסרת איבר ממחסנית ריקה מוביל למצב של שגיאה שנקראת <u>תת-גלישה</u> (Underflow). בפרוצדורה HASARA, אם המחסנית ריקה, הערך ELYON יהיה 0.

הפנייה לאיבר MACHSANIT[ELYON] מכוונת לאיבר MACHSANIT[0] שאיננו מוגדר וזו שגיאה. נוכל להתגבר על תקלה זו ע"י הגדרה של משתנה בוליאני שערכו יהפוך ל-TRUE כאשר מגיעים למצב של תת-גלישה.

```
PROCEDURE HASARA2 (VAR NATUN: SUG_NATUN;
                   VAR UNDERFLOW: BOOLEAN);
   BEGIN
     IF ELYON = 0 THEN
       UNDERFLOW := TRUE
     ELSE
       BEGIN
         UNDERFLOW:= FALSE;
         NATUN := MACHSANIT[ELYON];
         ELYON := ELYON - 1
       END
   END;
```

<u>16.3.2 יתרונות השימוש במחסנית</u>

נדגים את השימוש במחסנית באמצעות תכנית אשר קוראת רשימה של שמות וציונים של סטודנטים. הפלט הוא השם של הסטודנט, או הסטודנטים, שקבלו את הציון הגבוה ביותר.

כאשר משתמשים במערך רגיל לפתרון הבעיה הזו, הפתרון כולל שלושה צעדים:
א. כתיבה של השמות והציונים למערך.
ב. סריקה של המערך כדי לברר את הציון המקסימלי.
ג. סריקה של המערך פעם נוספת כדי לברר את השמות של כל הסטודנטים בעלי אותו ציון מקסימלי.

על ידי שימוש במחסנית אפשר להמנע מהחסנה של השמות במערך ומסריקתו פעמיים. מחזיקים במחסנית את השם של כל סטודנט שהציון שלו שווה לציון המקסימלי שזוהה עד עכשיו. כאשר מגלים סטודנט עם ציון גבוה יותר, מרוקנים את המחסנית וכותבים את השם של הסטודנט עם הציון המקסימלי המעודכן. תכנית שכתובה בשיטה זו ניתנת להלן:

```
PROGRAM TZIYUN_GAVOA (INPUT, OUTPUT);
CONST
  MAX_MACHSANIT := 20;
TYPE
  SUG_NATUN = ARRAY[1..20] OF CHAR;
VAR
  MACHSANIT: ARRAY[1..MAX_MACHSANIT] OF SUG_NATUN;
  ELYON: 0..MAX_MACHSANIT;
  SHEM : SUG_NATUN;
  TZIYUN, TZIYUN_GAVOA : INTEGER;
  GLISHA : BOOLEAN;
  PROCEDURE DECHIFA2 (NATUN: SUG_NATUN;
                      VAR  OVERFLOW: BOOLEAN); EXTERNAL;
  PROCEDURE HASARA2 (VAR NATUN: SUG_NATUN;
                     VAR UNDERFLOW: BOOLEAN); EXTERNAL;
```

```
PROCEDURE ROKEN;
BEGIN
  ELYON := 0
END;

PROCEDURE HADPES_MACHSANIT;
VAR
   I: INTEGER;
BEGIN
  FOR I:= 1 TO ELYON DO
     WRITELN (MACHSANIT[I])
END;

PROCEDURE KLITA;
BEGIN
  WRITELN ('כתוב שם וציון של סטודנט');
  READLN (SHEM, TZIYUN)
END;

BEGIN                                              (* תכנית ראשית *)
  ROKEN                                            (* רוקן את המחסנית *)
  KLITA;                                           (* קרא סטודנט ראשון *)
  TZIYUN_GAVOA:= TZIYUN;     (* טפל בסטודנט ראשון כאילו הציון
                                                   שלו הוא הגבוה ביותר. *)
  DECHIFA2 (SHEM, GLISHA);
  WHILE NOT (EOF OR GLISHA) DO
    BEGIN
      KLITA;
      IF TZIYUN > TZIYUN_GAVOA THEN
        BEGIN
          TZIYUN_GAVOA := TZIYUN;                  (* זהה מקסימום חדש *)
          ROKEN;                                   (* נקה את המחסנית *)
          DECHIFA2 (SHEM, GLISHA)
        END
      ELSE
        BEGIN
          IF TZIYUN = TZIYUN_GAVOA THEN
            DECHIFA (SHEM, GLISHA);
          IF GLISHA THEN
            WRITELN ('טעות. יש גלישה')
        END
    END;
  IF NOT GLISHA THEN
    HADPES_MACHSANIT
END.
```

<u>הסבר:</u>

השם הראשון מוכנס מיד למחסנית והציון הראשון נחשב לציון הגבוה ביותר. מכאן ואילך TZIYUN_GAVOA יוחלף רק אם יימצא ציון יותר גבוה מן הציון שנמצא ב-TZIYUN_GAVOA. במקרה כזה, מרוקנים את המחסנית. אם מוצאים ציון אשר שווה ל-TZIYUN_GAVOA דוחפים את השם של הסטודנט בעל ציון זה אל המחסנית.

16.4 תורים

תור (Queue) הוא רשימה שאפשר להוסיף לה ולמחוק ממנה איברים רק בתנאים אלה:

א. הוספות נעשות רק בסוף הרשימה.
ב. ביטולים נעשים רק בראש הרשימה.

שיטה זו נקראת FIFO (First In First Out) כלומר, סדר השרות הוא כסדר המופע. מכיון שההוספות הן תמיד בסוף הרשימה, יש צורך לדעת את מקום ההוספה של האיבר האחרון כדי להוסיף אחריו. דוגמא לתור FIFO הוא התור של אנשים העומדים לפני קופת הקולנוע כדי לקנות כרטיסים: אנשים מצטרפים לסוף התור ועוזבים אותו מראשו לאחר רכישת כרטיסים.

אפשר לייצג תור בשתי דרכים:

א. רשימה משורשרת עם רשימת מקומות פנויים.

לצורך זה דרושים שלושה משתנים:
RISHON - מצביע אל המקום של האיבר הראשון בתור,
ACHARON - מצביע אל המקום של האיבר האחרון בתור,
PANUY - מצביע אל המקום הראשון ברשימת המקומות הפנויים.

לניהול תור בשיטה זו דרושים שני מערכים: אחד עבור הנתונים ואחד עבור המצביעים.

ב. מערך אחד בלבד אשר מחזיק את נתוני המחכים בתור.
כאשר מגיעים למקום האחרון במערך ממשיכים אל המקום הראשון. מעדכנים באופן שוטף את המצביעים אל המקום הראשון ואל המקום האחרון בתור.

נחזור לדוגמא של התור לקופת הקולנוע כדי להדגים את שיטת הייצוג השנייה. נצהיר על המערך TOR ועל המשתנים המתלווים אליו.

```
CONST
   MAX_TOR = 10;   (* התור אינו יכול להכיל יותר מ-10 איברים *)
TYPE
   SUG_NATUN = PACKED ARRAY[1..25] OF CHAR;
VAR
   TOR: ARRAY [1..MAX_TOR] OF SUG_NATUN;
   RISHON, ACHARON: INTEGER;
```

במצב ההתחלתי יהיה: RISHON=1 ו-ACHARON=MAX_TOR. בדרך כלל, כאשר אדם מצטרף לתור מוסיפים אחד ל-ACHARON, כי מספר האיבר שמחזיק את האדם האחרון בתור גדל ב-1. רק אם ACHARON=MAX TOR (כמו במצב ההתחלתי) לא מוסיפים 1 ל-ACHARON, אלא משווים את ACHARON ל-1. נמצא, שאחרי שהאדם הראשון מצטרף לתור, גם RISHON וגם ACHARON שווים ל-1.

בנקודה מסוימת בזמן, התור יכול להיראות כך:

בנקודה מסוימת בזמן, התור יוכל להיראות כך:

TOR[1]	אברהם	האיבר הראשון במערך:
TOR[2]	יהודה	** האחרון בתור:
TOR[3]		
TOR[4]		
TOR[5]		
TOR[6]		
TOR[7]	חוה	** הראשון בתור:
TOR[8]	שרה	
TOR[9]	חנוך	האיבר האחרון במערך:

RISHON = 7   ** מצביע אל המקום הראשון בתור

ACHARON = 2   ** מצביע אל המקום האחרון בתור

16.4.1 כניסה ויציאה

את הכניסה ואת היציאה מן התור נבצע בעזרת שתי פרוצדורות, אשר נכנה אותן KENISA, ו-YETZIA בהתאמה.

כניסה לתור:

```
(1)  PROCEDURE  KENISA (NATUN: SUG_NATUN);
(2)      BEGIN
(3)              ACHARON:= ACHARON+1;
(4)              IF ACHARON > MAX_TOR  THEN
(5)                 ACHARON:= 1;
(6)              TOR[ACHARON]:= NATUN
(7)      END;
```

כאשר אדם נוסף מצטרףלתור, מוסיפים 1 למצביע אל המקום האחרון התפוס בתור. אם הערך החדש של ACHARON גדול ממספר המקומות במערך, מציבים 1 ב-ACHARON, ז"א חוזרים לתחילת המערך. אפשר להחליף את השורות (3)-(5) בפקודה אחת:

```
ACHARON := ACHARON MOD MAX_TOR + 1;
```

החיסרון בפקודה זו הוא בכך שהיא אינה מבליטה בבהירות את שני המקרים השונים שיכולים להתרחש: המקרה הטיפוסי, בו מוסיפים 1 ל- ACHARON והמקרה החריג, בו מחזירים את ערכו של ACHARON ל-1.

יציאה מתור:

```
(1)   PROCEDURE YETZIA (VAR NATUN: SUG_NATUN);
(2)   BEGIN
(3)     NATUN:= TOR[RISHON];
(4)     RISHON:= RISHON+1;
(5)     IF RISHON > MAX_TOR  THEN
(6)       RISHON:= 1
(7)   END;
```

בזמן יציאה מן התור, ראש התור נסוג לאחור מקום אחד, אל המקום שאחריו ולכן גם המציין המקביל במערך TOR גדל ב-1. היוצא מן הכלל הוא כאשר האדם הראשון בתור נמצא במקום האחרון במערך. כאשר אדם זה מקבל שירות ויוצא, הראשון בתור יהיה במקום הראשון במערך. רעיון זה מתבטא בשורות (4)-(6). גם בפרוצדורה זו ניתן לקצר ולהחליף שורות אלו בפקודה אחת:

```
RISHON := RISHON MOD MAX_TOR + 1;
```

כדי להגן מפני האפשרות של גלישה במערך שמחזיק את התור, כדאי להחזיק משתנה שיקרא MISPAR_BATOR, אשר ערכו ההתחלתי שווה ל-0. לפני שנאפשר כניסה נוספת נבדוק את ערכו של MISPAR_BATOR, ולא נאפשר את הכניסה אם ערכו שווה ל-MAX_TOR. כמו כן לא נרשה יציאה אם MISPAR_BATOR כבר שווה ל-0.

16.4.2 משטרי תור שונים

תור כפול (Double Ended Queue - DEQUE) הוא תור שמאפשר הוספות ויציאות כמו בתור רגיל ובנוסף, הוא מאפשר הוספות בראשו ויציאות מסופו.

כדוגמא לתור כפול ניקח את תור האנשים שמחכים במשרד של רופא. רוב האנשים מחכים לתורם כדי לקבל שירות. יש אנשים שנכנסים, בוחנים את התור ועוזבים מיד, מכיון שאינם רוצים לחכות. זהו מצב של יציאה מסוף התור.

אין אלו הדוגמאות היחידות לתורים. קיימים תורים עם אפשרות לסדרי עדיפות שונים. בדוגמא זו, אדם שיש לו עדיפות גבוהה משל אחרים יקבל שירות לפניהם, גם אם הצטרף אחריהם לתור. רק אם קדמו לו אחרים בעלי רמת עדיפות שווה לשלו, הוא ימתין ויקבל שירות לפי סדר המופע. כל זה חורג מהדיון בספר זה.

הפרוצדורות המתאימות לניהול תור כפול הן הרחבות של הפרוצדורות המתאימות למקרה של תור רגיל.

16.5 עצים בינריים (*)

לרשימה משורשרת יש יתרונות מבחינת ניהול איברי המערך, כי אפשר להוסיף ולמחוק ערכים מבלי להזיז את הערכים האחרים שנמצאים בה. הקושי הוא באפשרויות החיפוש. במערך מסודר רגיל יש שיטות מהירות למצוא איבר שמחזיק ערך מסוים. אפשר להשתמש למשל, בשיטת חיפוש בינרי (Binary search), אשר מנצלת את העובדה שהמערך ממויין. ברשימה משורשרת שיטה זו איננה מתאימה, כי אין קשר בין המקום הפיסי של איבר לבין המקום הלוגי שלו במערך.

עץ בינרי (Binary Tree) הוא סוג נוסף של רשימה משורשרת. האיבר הראשון ברשימה כזו הוא האיבר שמבחינה לוגית נמצא באמצע הרשימה. כל איבר יכול להיות משורשר לשני איברים אחרים, המכונים בן ימני (Right Child) או בן שמאלי (Left Child), בהתאמה. האיבר שמצביע אליהם נקרא אב או הורה (Father, או Parent).

הערך של איבר הבן הימני גדול בערכו מהערך של איבר האב והערך של איבר הבן השמאלי קטן בערכו מהערך של איבר האב. גם לאיבר הראשון וגם לבנים שלו לא חייבים להיות בנים, אבל אם יש, הכללים שהוזכרו תקפים, ז"א הבן הימני גדול בערכו מאביו והבן השמאלי קטן בערכו מאביו. אם יש רק בן אחד, הוא יחשב כימני או שמאלי בהתאם ליחס בינו לבין הערך של איבר האב.

כדוגמא של עץ בינרי נשתמש ברשימת החיות בגן החיות של הקיבוץ. בגן החיות

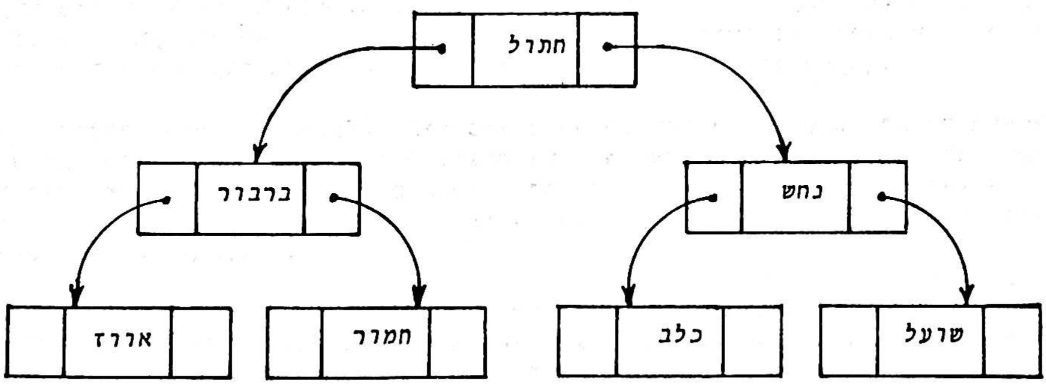
נמצאים כעת בעלי החיים ששמותיהם ממויינים לפי סדר א"ב עולה:

אווז	<---	(3)
ברבור	<---	(2)
חמור		
חתול	<---	(1)
כלב		
נחש		
שועל		

נניח שמחפשים "אווז" ורוצים למצוא אותו בשיטת חיפוש בינרי. מכיון שישנם שבעה איברים ברשימה, האיבר האמצעי הוא האיבר הרביעי, חתול. תחילה בודקים את האיבר אשר מסומן בספרה 1 ובו נמצא הערך חתול. היות והמלה חתול גדולה מהמלה אווז בסדר המילון, צריך להמשיך את החיפוש בחלק התחתון של הרשימה, באיזור הערכים הקטנים יותר. בשלב הבא מחלקים את קבוצת האיברים מאווז עד חתול ל-2. ישנם רק שלושה איברים בחלק זה והאמצעי בהם, ברבור, מסומן ב-2. מכיון שברבור גדול מאווז, לוקחים את קבוצת האיברים הקטנים יותר שבה נמצא רק איבר אחד, אווז, שהוא האיבר שמחפשים ומסומן 3 ברשימה.

בדוגמא זו נדרשו 3 צעדים, או השוואות, בכדי לסיים את החיפוש. מכיון שהמלה "אווז" הינה הראשונה ברשימה, היה נדרש צעד אחד בלבד כדי למצוא אותה בשיטת חיפוש לינארי. אך אם היה נדרש למצוא מלה אחרת, כמו נחש או שועל, צריך היה לבצע מספר רב של צעדים. בשיטת חיפוש בעץ בינרי אפשר למצוא מלים אלו במספר קטן יותר של צעדים. כאשר מספר האיברים ברשימה גדול, החיסכון במספר הצעדים למציאת ערך כלשהו הוא הרבה יותר משמעותי.

אפשר לרשום את שמות החיות במבנה של עץ בינרי, שבו כל מלבן שמחזיק שם של חיה נקרא צומת (Node). עורכים את הצמתים במבנה עץ בסדר שבו מבצעים חיפוש בינרי. ההבדל היחיד הוא בכך שחוצים את הרשימה בשני הכיוונים, כאשר החציון העליון הוא הבן הימני והחציון התחתון הוא הבן השמאלי.



כל צומת מורכב משלושה חלקים:

מצביע ימני	נתון	מצביע שמאלי

המצביע הימני של הצומת מורה על צמתים שמחזיקים ערכים אשר גדולים מאשר הערך שנמצא באותו צומת. המצביע השמאלי מורה על צמתים שערכם קטן מן הערך של צומת זה. הצומת שמכיל "חתול" הוא השורש (Root) של העץ. האיבר "חתול" הוא אב של האיברים "נחש" (בן ימני) ו"ברבור" (בן שמאלי). בדרך זו "שועל" הוא הבן הימני ו"כלב" הוא הבן השמאלי של "נחש", "ברבור" הוא האב של "חמור" ו"אווז". האיברים האחרונים בעץ הם אבות ללא בנים ומקובל לכנות אותם עלים (Leaves). עלה הוא איבר שאינו מצביע על איבר כלשהו.

בעץ זה אפשר להתקדם מן השורש והלאה שתי פעמים עד אשר מגיעים לעלה, ז"א עוברים דרך שלושה צמתים - השורש ועוד שניים. לכן קובעים שלעץ זה שלוש רמות (Levels). בזמן חיפוש משווים את האיבר בשורש עם האיבר שמחפשים. אם איבר השורש הוא זה שמחפשים, מסיימים את החיפוש. אם האיבר שמחפשים קטן מן השורש, ממשיכים בכיוון של המצביע השמאלי. הוא מורה אל תת-העץ בצד שמאל של השורש, אשר מורכב מאיברים שכולם קטנים בערכם מאיבר השורש. אם האיבר שמחפשים גדול מאיבר השורש, ממשיכים בכיוון של המצביע הימני אשר מורה אל האיברים הגדולים יותר.

עצי חיפוש בינריים הם מבנים רכורסיביים. כל תת-עץ של עץ חיפוש בינרי הוא בעצמו עץ חיפוש בינרי. כלומר, השורש גדול מכל האיברים שבצד שמאל וקטן מכל האיברים שבצד ימין שלו. בדוגמא זו, "ברבור" גדול מ"אווז" וגם מכל איבר שיכול להיות בן של "אווז".

שיטת בניית העץ גרמה לכך שכל איבר, למשל ברבור, גדול מן האיבר שהוא מצביע אליו משמאלו (אווז), קטן מן האלמנט שהוא מצביע אליו מימין (חמור) ועוד יותר קטן מן האלמנט המצביע אליו משמאלו (חתול). כדי למיין את העץ לפי סדר עולה צריך למיין כל תת-עץ משמאל לימין.

לסיכום, רשימה הינה עץ בינרי אם נוכל לומר לגבי כל צומת:

א. האיבר (אם יש) שהוא מצביע אליו משמאל קטן ממנו.
ב. האיבר (אם יש) שהוא מצביע אליו מימין גדול ממנו.

מחזיקים עץ בינרי במערך נתונים עם שני מערכי מצביעים שנקרא להם בהתאמה:
MATZBIA_YAMIN MATZBIA_SMOL

דוגמא לצורת ההחסנה של העץ הבינרי שהוזכר קודם:

SHORESH = 4

SUBSCRIPT	NETUNIM	MATZBIA_SMOL	MATZBIA_YAMIN
1	כלב	0	0
2	שועל	0	0
3	ברבור	6	7
4	חתול	3	5
5	נחש	1	2
6	אווז	0	0
7	חמור	0	0

כאשר מגיעים לסוף ענף, ערך המצביע הוא 0. לכן יש לעלה שני מצביעים שערכם הוא 0.

מייצגים עץ בשפת פסקל בעזרת הרשומה וההצהרות שמופיעות למטה:

```
CONST
    MAX_NETUNIM = 100;
TYPE
    MATZBIA = 0..MAX_NETUNIM;
    TZOMET = RECORD
                NATUN: SUG_NETUNIM;
                MATZBIA_SMOL: MATZBIA;
                MATZBIA_YAMIN: MATZBIA
             END;
VAR
    ETZ: ARRAY[1..MAX_NETUNIM]  OF  TZOMET;
```

במשתנה SHORESH נאחסן את מספר האיבר של הצומת, שהוא השורש של העץ. ערכו ההתחלתי יהיה 0 (Null). רשימת האיברים הפנויים תבנה כמו ברשימה משורשרת, כאשר PANUY יצביע על המקום הראשון ברשימה זו. רשימת האיברים הפנויים צריכה רק מצביע אחד, ונניח באופן שרירותי שהוא המצביע הימני.

נסקור עתה מספר פעולות אשר אפשר לבצע על עצים.

16.5.1 הוספת איברים לעץ

כדי להוסיף איבר לעץ קיים מבצעים תהליך של חיפוש הערך שלו. אם מוצאים ערך זה, אין צורך להוסיף אותו לעץ. אם לא מוצאים אותו, מגיעים לבסוף לאיבר שהמצביע שלו הוא 0. המקום שבו נרשום את האיבר החדש הוא מקום שעכשיו עוד לא קיים. כאשר נוסיף אותו, האיבר הקודם יצביע אליו.

נחזור לדוגמא של גן החיות. ארנבת הובאה לגן החיות ועל כן יש להוסיף את המלה "ארנבת" לרשימת החיות. מתחילים את החיפוש מן השורש. המלה "ארנבת" נמצאת בסדר האלפביתי לפני המלה "חתול" ולכן ממשיכים לחפש בכיוון של המצביע השמאלי, אל המלה "ברבור" (מספר 3 ברשימה). "ארנבת" קטנה גם ממנו, ולכן ממשיכים למלה "אווז" (6). "ארנבת" גדולה מ"אווז" ולכן נמשיך בכיוון ימין. המצביע הימני של אווז הוא 0 ולכן נוסיף כאן את המלה החדשה. נוכל לכתוב את הפקודות הבאות:

```
NETUNIM[8]:= ארנבת          (* הכנס את הערך החדש למקומו *)
MATZBIA_YAMIN[6];= 8        (* כוון את המצביע של השורש אל האלמנט החדש *)
MATZBIA_SMOL[8]:= 0         (* כוון את המצביע השמאלי של האלמנט החדש *)
MATZBIA_YAMIN[8]:= 0        (* כוון את המצביע הימני של האלמנט החדש *)
```

עץ נקרא <u>שלם</u> (Complete), אם כל העלים שלו הם באותה רמה. עץ נקרא <u>מאוזן</u> (Balanced), אם הפרשי רמה בין כל זוג עלים אינו עולה על 1.

לפני ההוספה של "ארנבת" העץ היה שלם ברמה 3. לאחר הוספת המלה "ארנבת", העץ איננו שלם אבל הוא עדיין מאוזן, כי יש לו איבר אחד ברמה הרביעית והשאר

הם ברמה 3. המספר המירבי של צמתים שיכולים להיות ברמה הרביעית הוא: $2^4-1=8$. ככל שמוסיפים יותר איברים יש יותר סיכויים שהעץ לא יהיה מאוזן.

נכתוב פרוצדורה אשר מקבלת כקלט שני ערכים:
א. KELET_CHADASH - איבר (ערך) שרוצים להוסיף לעץ,
ב. MAKOM - המקום של האיבר הראשון שבוחנים.
בסיום, המשתנה MAKOM מציין את המיקום של האיבר החדש שהוספנו. אם העץ כבר מכיל איבר עם הערך של KELET_CHADASH, המשתנה MAKOM יחזיר את המקום של איבר זה.

בכל שלב של התהליך, MAKOM הוא הצומת שעומדים לבדוק כדי לראות אם תוכנו שווה ל-NATUN_CHADASH. רק אם אף אחד מן הצמתים הקיימים לא מחזיק נתון זה, המשתנה MAKOM מקבל מקום של איבר שעד עכשיו לא היה מנוצל ואשר עתה נכתוב בו את NATUN_CHADASH.

בזמן הקריאה הראשונה לפרוצדורה, ערכו של MAKOM יהיה שווה לערך של SHORESH. הפרוצדורה רכורסיבית וקוראת לעצמה ולכן המשתנה MAKOM יקבל את ערכו בקריאה הבאה מן המצביע הימני או מן המצביע השמאלי של הצומת שבודקים בקריאה הנוכחית.

בכל קריאה לפרוצדורה מקבלים את הערך של המשתנה MAKOM מאחד מ-3 מקורות:
א. כערכו של SHORESH.
ב. כערכו של MATZBIA_YAMIN של צומת כלשהוא.
ג. כערכו של MATZBIA_SMOL של צומת כלשהוא.

הערך של MAKOM יהיה שווה ל-NULL בשלושה מקרים:
א. העץ ריק: הערך של SHORESH שווה NULL.
ב. מנסים להתקדם לימין, אבל המשתנה MATZBIA_YAMIN שווה NULL ולכן הוא מורה על כך שאין איברים נוספים באותו כיוון.
ג. מנסים להתקדם לשמאל, אבל המשתנה MATZBIA_SMOL שווה NULL, כי אין איברים נוספים באותו כיוון.

הקריאות הרכורסיביות מסתיימות באחד משני המקרים:
א. כאשר הערך של MAKOM שווה NULL. מוסיפים ערך זה לעץ.
ב. כאשר KELET_CHADASH שווה לאחד מן הערכים שנמצאים כבר בעץ. לא עושים דבר ולכן MAKOM יהיה שווה למיקום של אותו ערך.

להלן הפרוצדורה:

```
PROCEDURE  BENIYA (VAR MAKOM: MATZBIA;
                   KELET CHADASH: SUG NATUN);
BEGIN
  IF  MAKOM = NULL  THEN
                         (* הגענו לעלה ויש להוסיף איבר חדש *)
    BEGIN
                         (* מצאנו תא פנוי *)
      MAKOM:= PANUY;
                         (* עדכון רשימת המקומות הפנויים *)
      PANUY:= MATZBIA_YAMIN (PANUY);
                         (* הכנסת הערך לאיבר החדש *)
      ETZ[MAKOM].MATZBIA_SMOL:= NULL;
      ETZ[MAKOM].MATZBIA_YAMIN:= NULL
```

```
      END
    ELSE
                    הגענו למקום שתפוס ע"י אלמנט מסוים בעץ. *)
                    (* נבדוק עכשיו אם להתקדם לימין או לשמאל.
      BEGIN
        IF KELET_CHADASH < ETZ[MAKOM].NATUN  THEN
          BENIYA (ETZ[MAKOM].MATZBIA_SMOL, KELET_CHADASH);
        IF KELET_CHADASH > ETZ[MAKOM].NATUN  THEN
           BENIYA (ETZ[MAKOM].MATZBIA_YAMIN, KELET_CHADASH)
      END
  END;
```

אם הקלט החדש שווה לערך שנמצא באותו צומת, מסיימים פרוצדורה זו מכיון שאף אחת מפקודות IF לא מתבצעת. כתוצאה, מסתיימות כל הפרוצדורות שקראו לפרוצדורה זו. אפשר להוסיף לתכנית פקודה שתודיע למפעיל שמצב זה קרה, כדי שיוכל לבדוק את הסיבה ולתקן את הקלט, במידת הצורך.

אם נעקוב אחרי פקודות הפרוצדורה ונשתמש בנתונים חתול, נחש, כלב, שועל, ברבור, חמור ואווז, נקבל את העץ שהתווינו קודם. עץ זה הוא מאוזן וגם שלם. בצורה כללית אפשר לומר, שאם נקפיד להוסיף בנים רק אחרי אבות, העץ יהיה מאוזן. אך אם נקדים בנים לאבות, העץ לא יהיה מאוזן.

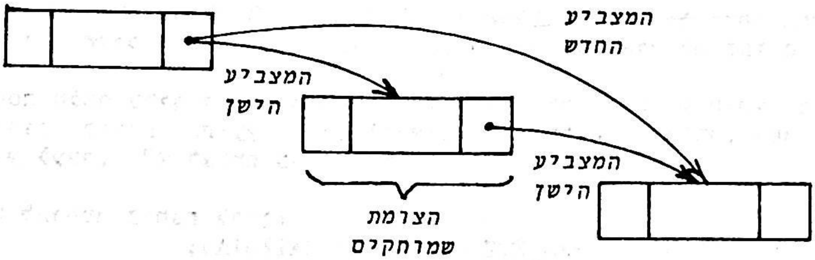
נניח שרוצים לבנות עץ מרשימה מסודרת. מוסיפים תחילה את האיבר האמצעי אשר חוצה את הרשימה לשתי קבוצות של ערכים גדולים וקטנים. אחר כך מוסיפים את האיבר האמצעי של קבוצת האיברים הגדולים ואת האיבר האמצעי של קבוצת הקטנים. ממשיכים בדרך זו עד שרושמים את כל האיברים. העץ הזה יהיה מאוזן במידה רבה.

בעית האיזון של עץ בינרי חורגת מדיון זה. הקורא המעוניין יפנה לביבליוגרפיה.

16.5.2 ביטול איברים מעץ

ביטול, או מחיקה של איברים מעץ קיים נעשית בדרך זו:

א. מוצאים את הצומת למחיקה.
ב. אם הצומת הוא עלה (ז"א שני המצביעים שלו שווים ל-0), משנים ל-"0" את המצביע אליו ומוסיפים את האיבר שהוא תופס במערך ETZ לרשימת המקומות הפנויים, כדי שאפשר יהיה להשתמש בו פעם נוספת.
ג. אם יש לצומת מצביע אחד שאינו 0, המחיקה מתבצעת בצורה דומה למחיקה של איבר מרשימה משורשרת רגילה. משנים את ערכו של המצביע אל הצומת שמוחקים כדי שיצביע אל הצומת שמקומו אחרי הצומת שמוחקים. בדרך זו ההורה צריך להצביע על נכדו לשעבר, שהפך כעת לבנו, כפי שמוצג בתרשים הבא:



ד. אם שני המצביעים של הצומת שרוצים למחוק שונים מ-0, צריך להעביר צומת ממקום אחר בעץ להחליף את הצומת שמוחקים, כפי שנראה בדוגמא הבאה.

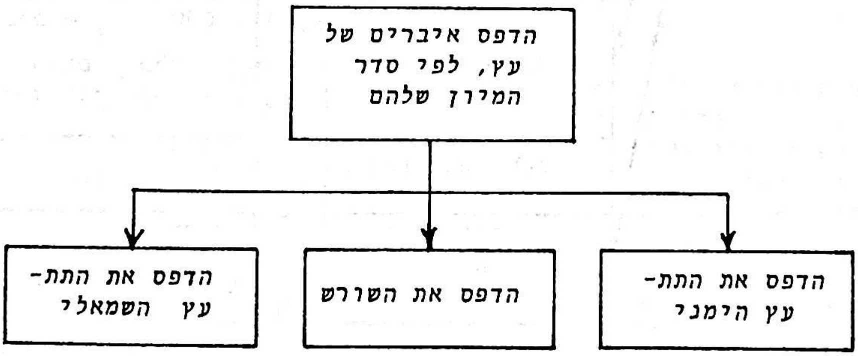
<u>דוגמא למחיקה:</u>

נניח שרוצים למחוק את השם "חתול" מרשימת גן החיות. במקומו יוכל לבוא איבר אחר, שגדול מכל האיברים בצד שמאל שלו וקטן מכל האיברים בצד ימין שלו. כדי למצוא איבר כזה, מתחילים לחפש מן האיבר שמוחקים ("חתול") לכיוון ימין כדי למצוא את האיבר הקטן ביותר, אשר קטן מכל איבר בצד ימין אבל גדול מ"חתול". בדרך זו נמצא את השם "כלב". אפשרות אחרת היא לחפש בצד שמאל את האיבר הגדול ביותר, אשר גדול מכל איבר אחר בצד שמאל אבל קטן מ"חתול". בדרך זו נמצא את השם "חמור". לאחר שבוחרים את האיבר המחליף ("כלב" או "חמור"), צריך למחוק אותו תחילה ממקומו הנוכחי. אחר-כך משנים את הערך של הנתון בצומת שהכיל את השם "חתול" כדי שיהיה עתה "חמור" או "כלב" בהתאם לאיבר המחליף שנבחר.

הכלל הוא שהאיבר שמחליף את האיבר המבוטל צריך להיות:
א. גדול מכל איבר אחר בצד שמאל.
ב. קטן מכל איבר אחר בצד ימין.
נשיג זאת ע"י בחירה של האיבר הקטן ביותר בצד ימין, או של האיבר הגדול ביותר בצד שמאל של האיבר המבוטל.

16.5.3 הדפסת עץ

נערוך תרשים היררכי להדפסת עץ:

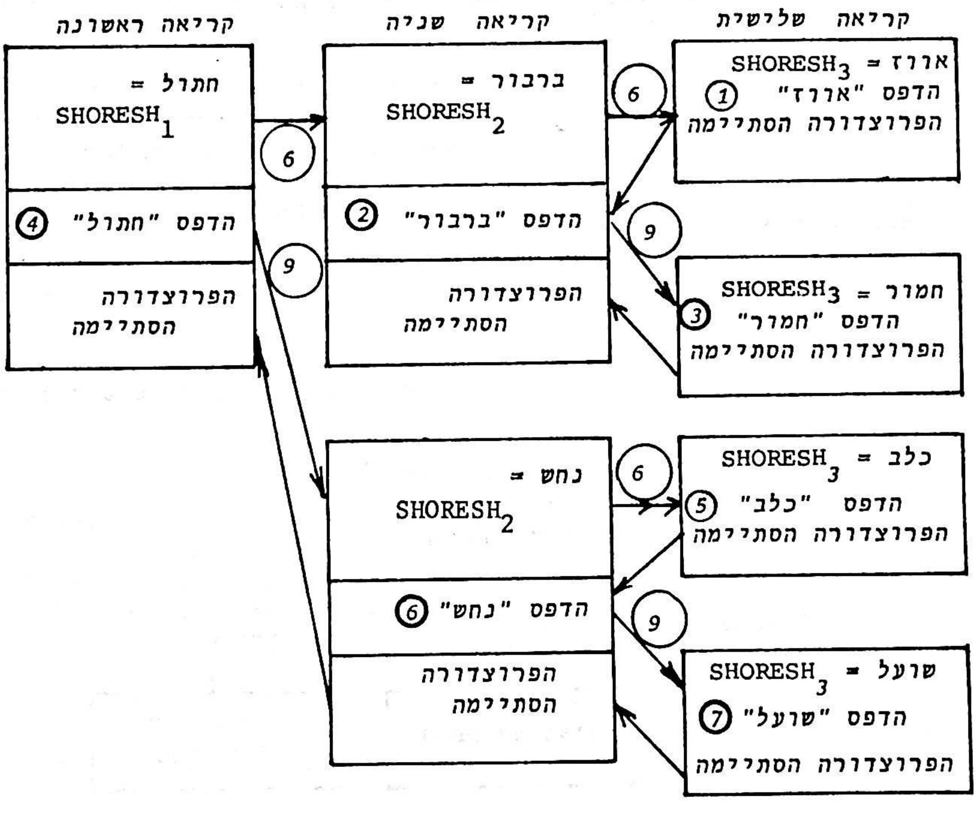


בעץ בינרי המצביע השמאלי מורה תמיד לאיבר שערכו יותר קטן והמצביע הימני מורה לאיבר שערכו יותר גדול. כדי לקבל הדפסה של האיברים לפי סדר המיון, סדר ההדפסה שלהם צריך להיות משמאל לימין. העובדה שגם לתת-עץ משמאל או מימין יכול להיות תת-עץ תגרום לביצוע חוזר של הפרוצדורה, אשר ייפסק רק כאשר נגיע לעלה, ז"א לצומת בלי בנים.

פרוצדורה להדפסה ניתנת להלן:

```
(1)   PROCEDURE HADPES_ETZ (SHORESH: MATZBIA);
(2)   BEGIN
(3)     WITH  ETZ[SHORESH]  DO
(4)       BEGIN
(5)          IF  MATZBIA_SMOL <> NULL  THEN
(6)            HADPES_ETZ (MATZBIA_SMOL);
(7)          WRITELN (NATUN);
(8)          IF  MATZBIA_YAMIN <> NULL  THEN
(9)            HADPES_ETZ (MATZBIA_YAMIN)
(10)      END
(11)  END;
```

מעקב אחרי פעולות הפונקציה הרכורסיבית בדוגמא זו מוצג בתרשים. המספר בתוך העיגול שליד כל חץ מורה על מספר הפקודה שיזמה את הקריאה הרקורסיבית. המספר, שבתוך המלבן ליד כל פקודת הדפסה, מציין את סדר ההדפסה.



סדר ההדפסה הוא לפי הסדר האלפביתי, כפי שנראה להלן:

- אווז
- ברבור
- חמור
- חתול
- כלב
- נחש
- שועל

16.6 תרגילים

16.6.1 תכניות

(1) כתוב תכנית אשר תבנה רשימה משורשרת, תאחזר מידע ותדפיס אותו.
הרשימה המשורשרת תכיל :
א. את השמות של חבריך לכיתה, מסודרים לפי סדר אלפביתי.
ב. ציון של הגשת 10 תרגילים ינתן במערך בן 10 איברים: האות X במקום i ($1 \le i \le 10$) תציין שהתלמיד השלים את התרגיל i בהצלחה והספרה אפס תציין כישלון.
ג. ציון מבחן ביניים בין 0 ל- 100.
ד. ציון מבחן סופי בין 0 ל- 100.
ה. ממוצע ציונים, אשר בנוי בצורה זו:
מבחן סופי * 0.5 + מבחן ביניים * 0.3 + מספר תרגילים * 2
ההוראות לאיחזור מידע מהרשימה הן בנות אות אחת בלבד, או אות ונתונים כמפורט להלן:
P - פלט של כל הרשימה, עם כותרות מתאימות.
T ושם תלמיד - פלט של רשומת תלמיד.
D ושם תלמיד - ביטול רשומת תלמיד מסוים.
H והשדות הדרושים להוספת רשומה (שם תלמיד, 10 תווי X או אפס כדי לציין הגשת תרגילים, 0 עד 3 ציונים) - הוספה של רשומת תלמיד.
Z וציון (בין 0 ל-100) - פלט של שם וציון ממוצע של כל תלמיד שיש לו ציון ממוצע גבוה מן הציון הנקוב בקלט.

(2) מטריצה דלילה מכילה אפסים רבים (בדרך כלל יותר מ-90%), למשל:

0	0	0	0	4	0	0	0
0	5	0	0	0	0	0	0
0	0	-3	0	0	0	0	0
0	0	0	0	0	0	0	0

כתוב תכנית שתקבל כקלט מטריצה בגודל M*N ותחזיק כל שורה כרשימה משורשרת. איברי הרשימה מכילים ערך חיוב או ערך שלילי ואת מספר העמודה שבה רשום אותו ערך. ערכים שהם 0 אינם מוחזקים ברשימה כדי לחסוך מקום. לפתרון הבעיה השתמש בווקטור עזר, ורשום בו באיבר i את מקומו של האלמנט הראשון בשורה i שברשימה המשורשרת.

התכנית צריכה להפיק את הפלטים הבאים:
א. הערכים שאינם אפס בכל שורה.
ב. המטריצה המקורית עם האפסים. להדפיס אותה ע"י שימוש ברשימה המשורשרת שבנית.

ג. הסכום של הריבועים של האלמנטים באלכסון מ-(1,1) עד (L,L), כאשר L הוא המינימום שבין M ו-N.

(3) אפשר לייצג פולינום כסדרה של זוגות מספרים. למשל, אפשר לייצג את הפולינום X^5+5X^3-16 כסדרה של הזוגות (-16,0), (5,3), (1,5), כאשר הערך הראשון בכל זוג הוא המקדם והשני הוא החזקה.

כתוב תכנית שמקבלת זוגות כאלו והופכת כל אחד מהם לרשימה משורשרת. בנה פרוצדורות שימצאו את הסכום והמכפלה של שני פולינומים שקוראים כקלט:

$5X^3-9X+5$

$X^2+9X+13$

התשובות תודפסנה כך:

הסכום הוא: 5X**3 + X**2 + 18

המכפלה היא: 5X**5 + 45X**4 + 56X**3 - 76X**2- 72X + 65

(4) בבניין מסוים יש 50 חדרים. בכל חדר ישנם מ-1 עד 5 אנשים וטלפון אחד. נתונה רשימה של כל האנשים בבניין כשהיא מסודרת בסדר אלפביתי, עם מספר הטלפון של כל אחד. רשימה זו תהיה הקלט לתכנית.

הפלט הוא רשימה של מספרי טלפון, ממויינים בסדר עולה. ליד כל מספר תוצג, בסדר אלפביתי, רשימת האנשים שיש להם אותו מספר טלפון. מומלץ לבנות רשימה משורשרת של האנשים שמשתמשים בכל טלפון. בכל צומת יהיו שני מפתחות: מספר הטלפון והשם. אפשר גם לבנות רשימה נפרדת בעבור כל מספר טלפון. רשימה נוספת יכולה להכיל את המיקום של הצומת הראשון ששייך לכל מספר טלפון.

(5) לחברת תעופה יש 5 טיסות ביום מסוים. הנתונים אודות טיסות אלו רשומים בטבלה:

מספר טיסה	תפושת המטוס
001	17
217	30
267	25
341	40
345	18

יש להגדיר שני מערכים:

א. TISOT - האיברים במערך זה הם רשומות בנות 3 שדות:
שדה ראשון: מספר הטיסה.
שדה שני: מספר המקומות הפנויים בטיסה.
שדה שלישי: ראש רשימה משורשרת שמחזיקה את שמות הנוסעים באותה טיסה.

ב. NOSIM - מערך שמכיל 5 רשומות משורשרות, כאשר המקום ההתחלתי של כל אחת מהן נמצא ב-TISOT. הרשומות במערך זה מכילות את שמות הנוסעים וכתובותיהם ומצביע אל הנוסע הבא ברשימה. הרשומות מסודרות בסדר אלפביתי.

המערכים NOSIM ו-TISOT והמשתנה PANUY הם משתנים גלובליים.

הקלט למערכת זו כולל 4 סוגי תנועות:

א. הזמנות - הקוד הוא HAZMANA. הנתונים הנילוים הם שם הנוסע ומספר הטיסה שלו.

ב. ביטולים - הקוד הוא BITUL. הנתונים הם שם הנוסע ומספר הטיסה שלו.

ג. בירור בקשר למספר המושבים הריקים - הקוד הוא MOSHAVIM. הנתון הוא מספר הטיסה.

ד. בירור של שמות הנוסעים בטיסה מסוימת - הקוד הוא HADPASA והנתון הוא מספר הטיסה.

מוצע להגדיר את הפרוצדורות הבאות, כאשר SHEM הוא שם נוסע ו-MISPAR הוא מספר טיסה:

HOSEF (SHEM, MISPAR) - הוספה של שם נוסע לטיסה מסוימת.

BATEL (SHEM, MISPAR) - ביטול הזמנה של נוסע לטיסה מסוימת.

BARER (MISPAR) - מספר המקומות הריקים בטיסה מסוימת.

HADPES (MISPAR) - הדפסה של רשימת נוסעים הרשומים לטיסה מסוימת.

רצוי לתכנת תחילה את הפרוצדורות HOSEF ו-HADPES ולבדוק אותם לבד. כאשר ידוע שפרוצדורות אלו פועלות כראוי והמערכת קולטת נתונים, יש לבדוק את הפרוצדורות BARER ו-BATEL.

(6) קרא רשימה של מספרים, רשום אותם במחסנית, הוצא אותם מהמחסנית והדפס לפי סדר ההוצאה מהמחסנית. המספרים יוצגו בסדר הפוך לסדר המקורי שלהם.

(7) נתונה סדרת מחרוזות מן הסוג SAS^T, לדוגמא:

++-A-++

+-+-A-+-+

S - מסמל סדרה של התווים + ו -

A - מסמל את האות A

ST - מסמל את הסדרה ש-S מסמל, אבל בסדר הפוך של התווים.

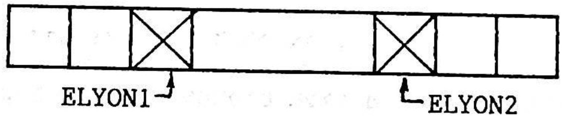
כתוב תכנית שבודקת בעזרת שימוש במחסנית אם המחרוזת אכן שייכת לסוג הזה.

(8) כתוב פרוצדורה:

```
HACHLAFA (VAR  NATUN: SUG_NATUN;
               MISPAR_EVAR: INTEGER)
```

הפרוצדורה מקבלת מספר איבר במחסנית A. היא מחזירה את הערך של הנתון שנמצא באיבר שמספרו במחסנית MISPAR_EVAR.

(9) אפשר להחזיק שתי מחסניות בווקטור אחד כפי שמוצג בתרשים:



כתוב את הפרוצדורות הבאות:

א. להוסיף נתון חדש למחסנית I:

DECHIFA (I: INTEGER; NATUN: SUG_NATUN)

ב. לבטל איבר ממחסנית מספר I:

HASARA (I: INTEGER; VAR NATUN: SUG_NATUN)

השתמש בפרוצדורות אלו כדי לבדוק אם מחרוזת נתונה היא מן הסוג SWS^T כאשר:

S הוא כמו שמתואר בתרגיל (7),
W מורכב ממחרוזת מן הסוג UVU^T:
U מורכב מסדרה של הסמלים * ו-/
V מסמל את האות B
U^T הוא סדרה הפוכה ל-U.

(10) הזמן שהוקצב לעיבוד של תכנית במחשב הוא שתי דקות. אם בסיום שתי הדקות לא הסתיים העיבוד, התכנית מפסיקה וחוזרת לסוף התור. אפשר להניח שתי קדימויות לפי סיווג העבודות:
- עבודות חשובות.
- עבודות לא חשובות.

נניח שנתונות העבודות הבאות:

מספר עבודה	זמן שירות	זמן המופע	קדימות
101	1	0	1
102	2	0	2
103	6	1	2
104	4	2	1
105	3	2	1
106	3	3	2
107	2	5	1

בנה שני תורים, אחד בשביל קדימות מסוג 1 ואחד בשביל קדימות מסוג 2. התור עם הקדימות הגבוהה תמיד מקבל שירות לפני התור השני. כתוב תכנית שתדפיס את מספר העבודה שנמצאת בשירות בסוף כל דקה, עד שהתורים מתרוקנים. חשב גם מספר ממוצע של תכניות בכל תור ב-21 הדקות הראשונות.

(11) (*) סופרמרקט מעוניין להשוות שתי חלופות להתקנה של דלפקי קופות. ההשוואה תבוצע באמצעות סימולציה שמטרתה להציג את שתי החלופות ולעזור למנהל הסופרמרקט להחליט איזו שיטה עדיפה. ציין זאת בסיום התרגיל.

שתי החלופות להשוואה הן:
א. 3 דלפקים רגילים.
ב. 2 דלפקים רגילים ודלפק אקספרס אחד, לאנשים שקנו 5 פריטים או יותר.
אפשר להניח שלקוחות מצטרפים לתור הקצר ביותר. אפשר גם להניח שכאשר אחד מן התורים לדלפקים מתפנה, מצטרפים אליו מן התורים האחרים.

נתונים:
א. המופע של הלקוחות לדלפקים הוא אקראי. הזמן הבין-מופעי הוא מספר אקראי בין 0 ל-360 שניות.
ב. מספר הפריטים שלקוח קונה הוא מספר אקראי בין 1 ל-40.
ג. משך השרות של הלקוח הוא 30 שניות כפול מספר הפריטים שהוא קונה.

כל אחד מהאירועים האלה קובע את האירוע הבא שהתכנית תטפל בו:
א. סיום שירות בכל אחד מן הדלפקים.
ב. מופע חדש של לקוח.

מבנה הפלט:

	דלפק 1	דלפק 2	A	סה"כ
מספר הלקוחות שקיבלו שירות במשך שעה				
זמן המתנה ממוצע ללקוח				
הזמן (%) שהדלפק פנוי				
מס' פריטים שטופלו במשך שעה				
אורך ממוצע של התור				

בסימולציה הראשונה האות A תוחלף בכותרת "דלפק 3" ובסימולציה השנייה היא תוחלף בכותרת "דלפק אקספרס".

(12) תור כפול מתיר כניסה ויציאה משני קצותיו.
כתוב פרוצדורות המאפשרות ניהול תור כפול והדפסת שמות האנשים אשר נמצאים בו. יש לציין מופע או יציאה מראשו או מסופו ואת שם האדם, אם האירוע הוא מופע.

נתוני הקלט יהיו שני תווים: התו הראשון יציין מופע או יציאה והתו השני יציין אם האירוע מתרחש בראש התור או בסופו. אם האירוע הוא מופע נצרף נתון שלישי: את השם.

נתון ראשון: מ - מופע נתון שני: ר - ראש התור
י - יציאה ס - סוף התור

להדפסה יש לציין את הדרישה בלבד: "ה".
נניח שנתון הקלט הזה:

מ ר יעקב
מ ר טובה
מ ס שמריהו
י ר
מ ר חניתה
ה
י ס
מ ס חנה
מ ס אהרון
ה

דוגמת פלט תהיה:

תור כפול = חניתה
יעקב
שמריהו
תור כפול = חניתה
יעקב
חנה
אהרון

(13) (*) כתוב פרוצדורות לבצע את הפעולות הבאות:
א. למחוק איברים מעץ.
ב. להדפיס תוכן של עץ לפי סדר מיון עולה.

(14) (*) כתוב תכנית שקוראת פיסקה ומכניסה בצורה סידרתית את כל המלים לתוך עץ בינרי, כשהן מסודרות בעץ לפי סדר אלפביתי. בכל צומת של העץ רשומים המלה ומספר המופעים שלה. כתוב תכנית שתדפיס את רשימת המלים בסדר אלפביתי וליד כל מלה - את מספר המופעים שלה.

(15) (*) בחנות לממכר חלקי חילוף רוצים לבנות עץ פריטים שיהיה מסודר לפי מספר פעולה. כל צומת יכלול מספר פעולה, תאריך, שם החלק שנמכר ושם לקוח. נתונים לדוגמא הם:

מס' פעולה	תאריך	חלק	לקוח
1	2/10	A	פרידמן
2	2/10	B	אברהם
3	3/10	A	רחמים
4	4/10	B	זהבי
5	4/10	C	רחמים
6	5/10	A	יחיאל
7	7/10	D	יחיאל
8	7/10	C	יחיאל
9	7/10	A	רחמים
10	8/10	B	שבתאי

כתוב פרוצדורה כדי להפוך את הרשימה המסודרת של נתוני הקלט לעץ מאוזן ולהדפיס אותו לפי סדר הפעולות.
רמז: כתוב פרוצדורה רקורסיבית ETZ(KATAN,GADOL) שתשבץ את הערך האמצעי שבין KATAN ל-GADOL. הפרוצדורה תקרא לעצמה פעמיים באופן רקורסיבי כדי לכתוב את הערך האמצעי בתחום העליון ואת הערך האמצעי בתחום התחתון ותמשיך בדרך זו, עד אשר הערך של GADOL קטן מן הערך של KATAN.

הצג את כל הפעולות בהן כלול חלק A ואת כל הפעולות ללקוח בשם רחמים.

(16) (*) חזור על תרגיל מס' 5 כאשר שמות הנוסעים מוחזקים בעץ בינרי.

פרק 17

מבני נתונים דינמיים

17.1 הגדרה ויישומים

מבנה נתונים דינמי (Dynamic Data Structure) הוא מבנה נתונים שגודלו אינו קבוע מראש ומספר איבריו משתנה.

רשימה משורשרת (ראה פרק 16), אפשר להחזיק במערך שמספר האיברים שלו קבוע, אשר נוכל לכנות אותו מבנה נתונים סטטי (Static Data Structure). כאשר הרשימה גדלה או קטנה, מספר האיברים שבשימוש גדל או קטן בהתאמה, אבל גודל המערך אינו משתנה. בנוסף לזה, אם גודל הרשימה עולה על מספר האיברים במערך, מגיעים למצב של גלישה.

החסרונות של רשימה משורשרת במבנה נתונים סטטי המוחזק במערך הם שניים:

א. מחזיקים בזיכרון מקומות פנויים כדי שיהיו זמינים בעת הצורך. גם כאשר מקומות אלה אינם מנוצלים, אי אפשר לפנות את הזיכרון למטרות אחרות.
ב. כאשר יש דרישה למספר איברים אשר גבוה ממספר האיברים במערך, מגיעים למצב של גלישה.

כדי להתגבר על חסרונות אלה משתמשים בעת הצורך במשתנה-מצביע (Pointer Variable), אשר מאפשר ליצור מבנה נתונים דינמי.

משתנה-מצביע שונה ממצביע רגיל שהזכרנו בדיון על רשימות משורשרות ועצים רק בדבר אחד:

- המצביע הרגיל הוא מציין (Subscript), או מספר סידורי, של אחד מן האברים במערך של הרשימה המשורשרת, אשר הוצהר בראש התכנית.
- משתנה-מצביע הוא כתובת של מקום פנוי בזיכרון, שלא הוצהר תחילה כחלק ממערך, אשר בו כותבים את האיבר החדש.

הכתובת, אשר המשתנה-מצביע מורה אליה לא קשורה כלל אל התכנית. לכן, כדי להוסיף איבר לרשימה במבנה נתונים דינמי, אין פונים אל רשימת פנויים שהוכנה מראש, אלא אל מאגר של מקומות פנויים בזיכרון. המקומות במאגר זה פנויים לשימוש של כל תכנית עד להקצאה שלהם.

הפנייה למאגר נעשית בעזרת הפונקציה: NEW (ארגומנט)
שתכונותיה הם:

א. הפונקציה NEW מבקשת מקום בזיכרון שבו אפשר יהיה להחזיק איבר אחד מן הסוג הרצוי. הכתובת של מקום זה נמצאת במשתנה-מצביע שהוא ה"ארגומנט" של הפונקציה.

ב. אם הסוג של האיבר הוא רשומה, הפונקציה NEW מאפשרת להתיחס לשדה מסוים ברשומה זו בעזרת הכינוי: שם-שדה.^משתנה-מצביע

ג. הפונקציה NEW מאפשרת לפנות לאיבר כולו ע"י השם: ^משתנה-מצביע

המזהה "משתנה-מצביע" מתייחס לכתובת של האיבר בשטח המאגר בזיכרון ולא לערך שלו. הציון של החץ מימין לשם המשתנה-מצביע מורה על כך שמתיחסים לערכו, ז"א לתוכן של האיבר שנמצא באותה כתובת. במשתנה רגיל, שאינו משתנה-מצביע, פונים תמיד לתוכן של המשתנה ולא לכתובת שלו ולכן אין צורך להבדיל ביניהם.

דוגמא להגדרה של משתנה-מצביע:

```
 (1)  TYPE
 (2)    MATZBIA = ^TZOMET;                  (* סוג מצביע *)
 (3)    TZOMET = RECORD
 (4)               SADE1: SUG1;
 (5)               SADE2: SUG2;
 (6)               SADEn: SUGn;
 (7)               KESHER: MATZBIA
 (8)             END;
 (9)  VAR
(10)    P,Q : MATZBIA;
```

<u>הסבר:</u>

(2) המלה MATZBIA היא השם של הסוג החדש שמגדירים, "סוג מצביע". המהדר מזהה זאת על-פי החץ שבצד שמאל של השם של הרשומה TZOMET, אשר בשלב זה הוא אינו יודע את גודלה. זהו המקרה היחיד בכתיבת תכנית בפסקל שמותר להשתמש בשם של סוג, או של משתנה בצד ימין של פקודה או הצהרה כאשר הגדרתו אינה ידועה.

משתנה שסוגו MATZBIA אינו מחזיק ערך, אלא כתובת, שגודלה 2 או 4 בתים, בלי קשר לגודלה, או להרכבה של הרשומה TZOMET. לכן ההגדרה של הרשומה אינה חשובה בעת הקצאת מקום לכתובת שלה. לאחר הידור ובזמן הביצוע של התכנית והפונקציה NEW, צריך לדעת את גודלה של הרשומה TZOMET לשם בקשת הקצאה מהמאגר הפנוי. בשלב זה ידוע כבר הגודל של הרשומה מסוג TZOMET.

(7) לאחר שמצהירים על הסוג MATZBIA, משתמשים בו כדי לתאר את השדה KESHER בתוך הרשומה TZOMET. שדה זה מחזיק מצביע לאיבר הבא ברשימה משורשרת. השימוש בסוג MATZBIA בצד ימין של הביטוי אפשרי כעת מכיון שהוא הוצהר בשורה (2), כי אסור להשתמש בשם משתנה או סוג שאינו מוגדר.

השם MATZBIA מופיע בשורה (2), (7) ו-(10) אבל הוא לא מופיע בשום מקום אחר בתכנית. המידע הקשור לשם MATZBIA נמצא בצד ימין של הביטוי בשורה (2)

בלבד. אפשר להשמיט שורה (2) ולכתוב במקום המלה MATZBIA את הסימון ^TZOMET בשורות (7) ו-(10) כדי לקבל את ההצהרה הבאה:

```
TYPE
  TZOMET = RECORD
             SADE1: SUG1;
             SADE2: SUG2;
             SADEn: SUGn;
             KESHER: ^TZOMET
           END;
VAR
  P,Q: ^TZOMET;
```

התוכן של המשתנה P לא מוגדר עד שרושמים בו ערך. הרישום יכול להיעשות באחת משלוש דרכים:

א. הפעלת הפונקציה NEW(P). פונקציה זו מעמידה לרשות התכנית את המקום הדרוש ושמה את <u>הכתובת</u> של אותו מקום במשתנה P.

ב. השוואה ל-NIL. NIL היא מלה שמורה אשר מציינת ערך מוגדר מראש. ומשמעותה, שהמצביע עדיין איננו מופעל כי הוא מחכה לקבל כתובת. השוואת המצביע ל-NIL נעשית ע"י פקודת הצבה רגילה: P^.KESHER:= NIL
(* מציבים NIL במצביע שהוא חלק מהרשומה מסוג TZOMET *)
או כותבים: P:= NIL
(* מציבים NIL במשתנה המצביע P בעצמו *)
בהצהרות אלו יש שני משתנים-מצביעים נפרדים: אחד בתוך הרשומה (P^.KESHER) שתפקידו יהיה להצביע אל האיבר הבא ברשימה, ואחד שמצביע על הרשומה הזאת ממש (P).

ג. השוואה של P למשתנה-מצביע אחר (למשל בפקודת הצבה P:= P^.KESHER) בין אם ערכו NIL ובין אם הוא מכיל כתובת של ממש.

<u>17.2 רשימות משורשרות</u>

אפשר לתאר רשימה משורשרת בעזרת מבנה נתונים דינמי. נשתמש בדוגמא של בנייה והדפסה של רשימה כזו. שיטה זו איננה מחייבת להקצות מראש מקום למערך, מכיוון שאפשר להקצות מקום לאיבר נוסף בכל פעם שדרוש.

```
(1)     PROGRAM RESHIMA (INPUT,OUTPUT);
(2)     TYPE
(3)       MACHROZET= PACKED ARRAY [1..10] OF  CHAR;
(4)       MATZBIA= ^TZOMET;
(5)       TZOMET= RECORD
(6)                 SHEM: MACHROZET;
(7)                 HABA: MATZBIA
(8)               END;
(9)     VAR
(10)          ROSH, P, Q: MATZBIA;
```

```
(11)        PROCEDURE  KRA_SHEM (VAR N:MACHROZET);
(12)        BEGIN
(13)          WRITELN ('   תן שם');
(14)          READ (N);
(15)          READLN
(16)        END;

(17)     BEGIN                (* RESHIMA *)
(18)        NEW(P);
(19)        ROSH:= P;
(20)        KRA_SHEM (P^.SHEM);               (* האיבר הראשון ברשימה *)
(21)        P^.HABA:= NIL;
(22)        WHILE  NOT EOF DO                 (* הוספת איברים לרשימה *)
(23)           BEGIN
(24)             Q:=P;
(25)             NEW(P);
(26)             KRA_SHEM (P^.SHEM);
(27)             Q^.HABA:= P;
(28)             P^.HABA:= NIL
(29)           END;
(30)        P:= ROSH;         (* הדפסת תוכן הרשימה לפי סדר האיברים *)
(31)        WHILE  P <> NIL  DO
(32)           BEGIN
(33)             WRITELN (P^.SHEM);
(34)             P:= P^.HABA
(35)           END
(36)     END.
```

<u>הסבר:</u>

(21)-(18) בניית הצומת הראשון. לאחר שיוצרים את המקום עבור האיבר הראשון בשורה (18), רושמים את כתובתו במשתנה-מצביע ROSH (19) וקוראים לתוך הרשומה ^P את השם הראשון שברשימה (שורה 20). היות ואין איברים אחרים ברשימה בשלב הזה, מציבים NIL בשדה HABA ברשומה ^P שמצביע על האיבר הבא (שורה 21).

(29)-(22) בפקודות אלו שומרים את הכתובת של האיבר האחרון שהוספנו (24), יוצרים איבר חדש (25) ורושמים בו את השם החדש (26). אח"כ חוזרים לאיבר הקודם ורושמים בשדה המצביע שלו את הכתובת של האיבר שעתה הוספנו, שורה (27). לבסוף רושמים NIL בשדה המצביע של האיבר החדש, שורה (28). תהליך זה נמשך עד שמסיימים את הטיפול בכל הנתונים.

(35)-(30) בשורות אלו מדפיסים את כל הרשימה, החל באיבר הראשון. רושמים את הכתובת של האיבר הראשון, ROSH, בתוך המצביע P ומדפיסים את השם שנמצא באותה כתובת (שורה 33). אח"כ מעבירים ל-P את הכתובת של האיבר הבא (34) וחוזרים על פעולת הכתיבה (33) עד שמגיעים לאיבר האחרון (P=NIL).

תדפיס הרצה של התכנית:

קלט:

```
תן שם^^^
AVRAHAM
תן שם^^^
YITZHAK
תן שם^^^
YAAKOV ^Z          (* Z^ מסמל סוף קלט *)
```

פלט:

```
AVRAHAM
YITZHAK
YAAKOV
```

כאשר משתמשים במצביעים אין צורך להגדיר מערך שיכול להחזיק את המספר המירבי של האיברים, כי משתמשים בזיכרון לפי מספר הכניסות ברשימה שבונים. בדוגמא זו הוספנו את האיברים בסוף הרשימה לפי סדר הגעתם ולא ניצלנו את היתרון של רשימה משורשרת להוספת איברים במקום כלשהו ברשימה, כדי ליצור מיד רשימה ממויינת.

17.2.1 הוספה וביטול של איברים ברשימה מסודרת

הוספה של איברים לרשימה משורשרת יכולה להיות באחד מתוך שלושה מקומות ברשימה:

א. ראש הרשימה: כאשר מוסיפים איבר לפני האיבר הראשון ברשימה צריך לעדכן את המשתנה ROSH המצביע על ראש הרשימה. למשל, הוספה למחסנית (ראה בהמשך). דוגמא אחרת היא הוספה לרשימה מסודרת, כאשר האיבר החדש קטן מכל אלה שנמצאים כבר ברשימה.

ב. בין שני איברים: אם מוסיפים את הרשומה R^ בין הרשומות Q^ ו-P^, צריך לכוון את המצביע ב-Q^ כדי שיצביע על R^ (Q^.HABA:=R) ואת המצביע ב-R^ שיצביע על P^ (R^.HABA:=P). לדוגמא, הוספה לרשימה מסודרת.

ג. סוף הרשימה: איבר חדש ירשם אחרי האיבר האחרון ברשימה ואז המצביע של האיבר החדש יהיה תמיד שוה ל-NIL. לדוגמא, הצטרפות של לקוחות בסוף תור שמחכה לשרות. דוגמא אחרת היא הוספה לרשימה מסודרת כאשר האיבר החדש גדול מכל אלה שנמצאים כבר ברשימה.

ביטול של איברים מרשימה משורשרת יכול להיות באותם שלושה מקומות שהוזכרו קודם. יש לעדכן את המצביעים כנדרש בכל ביטול.

התכנית הבאה מדגימה הוספה וביטול של איברים ממקומות שונים ברשימה. בונים תחילה רשימה מסודרת ואח"כ משמיטים ממנה איברים שונים לפי הוראת המפעיל.

```
PROGRAM HOSAFA_MECHIKA (INPUT,OUTPUT);
TYPE
     TAV10 = PACKED ARRAY [1..10] OF  CHAR;
     MATZBIA= ^TZOMET;
     TZOMET= RECORD
                  SHEM: TAV10;
```

```
                 HABA: MATZBIA
               END;
VAR
   ROSH, P, Q: MATZBIA;

PROCEDURE KRA_SHEM (VAR NAME : TAV10);     (* קריאת נתונים *)
BEGIN
  WRITELN ('כתוב שם להוספה או לביטול');
  READ (NAME);
  READLN
END;

PROCEDURE  HADPES_RESHIMA;                          (* הדפסה *)
BEGIN
  P:= ROSH;
  WHILE P <> NIL DO
   BEGIN
     WRITELN (P^.SHEM);
     P:= P^.SHEM
   END
END;

PROCEDURE HOSAFA;                             (* הוספת איברים *)
VAR
    R: MATZBIA;
BEGIN
    NEW(R);
    KRA_SHEM (R^.SHEM);
    Q:= NIL;
    P:= ROSH;
    WHILE (P <> NIL) AND (P^.SHEM <= R^.SHEM) DO
      BEGIN
        Q:= P;
        P:= P^.HABA
      END;
    IF  Q= NIL THEN          (* המקום של האיבר החדש נמצא *)
      BEGIN                               (* בראש הרשימה *)
        ROSH:= R;
        R^.HABA:= P
      END
    ELSE
      IF  P <> NIL  THEN     (* המקום של האיבר החדש נמצא *)
        BEGIN                             (* בתוך הרשימה *)
          Q^.HABA:= R;
          R^.HABA:= P
        END
      ELSE
        BEGIN                (* המקום של האיבר החדש נמצא *)
```

```
            Q^.HABA:= R;                           (* בסוף הרשימה *)
            R^.HABA:= NIL
          END
 END;                       (* HOSAFA *)

  PROCEDURE MECHIKA;        (* ביטול איברים *)
  VAR
    X: TAV10;
  BEGIN
    KRA_SHEM (X);
    Q:= NIL;
    P:= ROSH;
    WHILE (P <> NIL) AND (P^.SHEM < X) DO
      BEGIN
        Q:= P;
        P:= P^.HABA
      END;
    IF P <> NIL THEN
      IF P^.SHEM= X THEN
        BEGIN
          IF Q= NIL THEN
            ROSH:= P^.HABA
          ELSE
             Q^.HABA:= P^.HABA;
          DISPOSE (P)                (* ראה הסבר בהמשך *)
        END
      ELSE
        WRITELN (X, ' לא מצאנו את השם')
    ELSE
      WRITELN (X, ' לא מצאנו את השם')
  END;                  (* MECHIKA *)

  BEGIN                                            (* התכנית הראשית *)
    ROSH:= NIL;
    WRITELN ('בניית רשימה משורשרת');
    WRITELN;
    WHILE NOT EOF DO
      HOSAFA;
    RESET (INPUT);
    HADPES_RESHIMA;
    WRITELN;
    WRITELN  ('איברים שמבטלים מן הרשימה');
    WRITELN;
    WHILE NOT EOF DO
       MECHIKA;
    HADPES_RESHIMA
  END.
```

הסבר:

(16)-(11) פרוצדורה זו נקראת ע"י הפרוצדורות HOSAFA ו-MECHIKA כדי לקרוא שם מן המסך להוספה או לביטול. מניחים שמותר לקרוא את הנתון, אשר מוגדר ע"י PACKED ARRAY, בפקודת קריאה אחת. מהדר שלא מאפשר קריאה כזו ידרוש לולאה שקוראת כל תו בנפרד.

(25)-(17) פרוצדורה זו עוברת על פני הרשימה מראשה ועד האיבר האחרון, אשר קיים עבורו P^.HABA=NIL ומדפיסה כל שם שמופיע בה.

(55)-(26) פרוצדורה זו יוצרת צומת חדש R^ (שורה 30) וקוראת לתוכו שם חדש (שורה 31). אח"כ היא עוברת על פני הרשימה עד שהיא מוצאת איבר אשר יותר גדול מן האיבר החדש, או עד שהיא מגיעה לסוף הרשימה. בגמר ביצוע הלולאה בשורות (34)—(38) קיימות שלוש אפשרויות למצבו של האיבר החדש:

א. האיבר החדש קטן מכל האיברים שנמצאים כבר ברשימה. במצב זה המשתנה Q שווה לערך ההתחלתי שלו, NIL (39) כי לא נכנסנו כלל אל הלולאה (כי R^.SHEM < P^.SHEM). כתוצאה, R יהפוך להיות הכתובת של התחלת הרשימה (41) ויצביע על האיבר שהיה קודם בראש הרשימה (42).
ב. האיבר החדש קטן מאחד מאיברי הרשימה, אבל לא מכולם. אם זה קורה, Q<>NIL וגם P<>NIL. במקרה כזה מציבים במצביע של האיבר הקודם Q^ את הכתובת של R^ (47) ובמצביע באיבר R^ מציבים את הכתובת של P^ (48).
ג. האיבר החדש גדול מכל איברי הרשימה. מצב זה קיים אם יוצאים מן הלולאה בשורות (34)-(38) לא בגלל שקיים R^.SHEM<P^.SHEM, אלא מכיון שקיים P^.HABA=NIL. הלולאה ממשיכה עד אשר היא מגיעה לאיבר האחרון ברשימה, שכתובתו נמצאת ב-Q (36). במקרה כזה משנים את המצביע של Q מ-NIL ל-R (52) ורושמים NIL במצביע של R^, שהוא עכשיו האיבר האחרון.

(81)-(56) פרוצדורה שמבטלת איברים מהרשימה. עושים לולאה ומחזיקים את האיבר הקודם ב-Q. הלולאה מסתיימת כאשר X הופך להיות שווה או קטן מאחד האיברים ברשימה. מסיימים את הלולאה הזו באחד מהתנאים הבאים:

א. שורה (80): P=NIL. הגענו לסוף הרשימה ולא היה איבר גדול או שווה ל-X. מדפיסים הודעה שהשם לא נמצא ברשימה.
ב. שורה (69): P<>NIL, P^.SHEM=X ושורה (71): Q=NIL. השם שרוצים לבטל הוא הראשון ברשימה. העובדה ש-Q נשאר שווה ל-NIL מורה על-כך שלא נכנסנו כלל ללולאה. לכן משנים את ראש הרשימה מ-P ל: P^.HABA (72).
ג. שורה (73): P<>NIL, P^.SHEM=X, Q<>NIL. מצאנו את השם שמחפשים, אך הוא איננו בראש הרשימה (Q כבר איננו שווה ל-NIL). לא צריך לעדכן את ROSH, אלא רק את המצביע באיבר Q^ שלפניו כך, שיצביע על האיבר שאחרי P, שכתובתו P^.HABA ולא על האיבר P בעצמו.

במקרים ב' ו-ג' מצאנו את האיבר שרוצים לבטל. הביטול נעשה ע"י העברה של האיבר P חזרה למאגר האיברים הפנויים, ע"י שימוש בפונקציה חדשה, DISPOSE(P) (75).
הצורה הכללית של הפונקציה היא: DISPOSE (ארגומנט)
"ארגומנט" הוא השם של משתנה-מצביע שערכו הנוכחי הוא הכתובת של האיבר שאת מקומו רוצים להחזיר עכשיו למאגר.

ד. שורה (78): P<>NIL, X<P^.SHEM. מצב זה קורה אם השם למחיקה קטן מן האיבר הראשון, או שהוא נמצא בין שני איברים אחרים ברשימה. בשני המקרים שם זה איננו מופיע ברשימה ויש לדווח על שגיאה.

(96)-(82) התכנית הראשית. מוסיפים שמות לרשימה עד שמגיעים לסוף הקלט. בנקודה זו חייבים להפוך את הפונקציה הבוליאנית EOF מ-TRUE ל-FALSE כדי שנוכל להוסיף ולקרוא נתוני קלט מן המסך בהמשך התכנית. זאת עושים בשורה (88). פקודת RESET(INPUT) פותחת את קובץ INPUT (לוח המקשים) מחדש ומעבירה ערך FALSE ל-EOF.

17.3 מחסניות

צומת במחסנית (Stack) מוגדר כמו צומת ברשימה משורשרת וההבדל ביניהם מתבטא רק ביחס לפעולות שמפעילים עליהם. במקרה של מחסנית הטיפול יותר פשוט, כי כל ההוספות והביטולים מוגבלים לראש הרשימה.

הצהרה של מחסנית:

```
TYPE
    MATZBIA= ^MACHSANIT;
    MACHSANIT= RECORD
                  NATUN: SUG_NATUN;
                  HABA: MATZBIA
               END;
```

פרוצדורה כדי להוסיף איברים למחסנית (Push):

```
PROCEDURE DECHIFA (X: SUG_NATUN; VAR ELYON: MAZBIA);
                              (* הוסף איבר X לראש המחסנית *)
 VAR
    P: MATZBIA;
 BEGIN
    NEW(P);                   (* צומת חדש *)
    P^.NATUN:= X;             (* אחסן את הנתון *)
    P^.HABA:= ELYON;  (* האיבר החדש יצביע על האיבר
                      שהיה קודם בראש הרשימה. *)
    ELYON:= P          (* האיבר החדש הופך להיות
                      בראש המחסנית. *)
 END;
```

פרוצדורה כדי להסיר איברים מן המחסנית (Pop):

```
PROCEDURE HASARA (VAR X: SUG_NATUN; VAR ELYON: MATZBIA);
                              (* הסר את האיבר מראש המחסנית *)
 VAR
   P: MATZBIA;
 BEGIN
   X:= ELYON^.NATUN;
   P:= ELYON;            (* אחסן ב-P את המצביע לאיבר העליון. *)
```

```
      (* העבר את הכתובת של הנתון העליון החדש אל המצביע ELYON. *)

  ELYON:= ELYON^.HABA;
        (* האיבר העליון החדש הוא האיבר שהיה קודם האיבר השני *)
  DISPOSE(P)
END;
```

17.4 תורים

אפשר לייצג איברים בתור (Queue) בעזרת מצביעים. המתכנת איננו צריך לטפל באיברים פנויים כאשר הוא עובד עם מצביעים, כפי שראינו גם בייצוג של רשימה משורשרת ובמחסנית.

בייצוג תור ע"י רשימה משורשרת, כדאי לשים שני מצביעים בכל צומת, אחד שמצביע לפנים ואחד שמצביע לאחור. כדאי להשתמש גם בשני מצביעים חיצוניים, אחד שמצביע אל ראש התור ואחד שמצביע אל סופו. מצביעים אלה יאפשרו לגשת מיידית לראש התור כדי לבטל את האיברים שגמרו לקבל שרות, או לגשת לסוף התור כדי להוסיף איברים חדשים.

נדגים את ניהול התור ע"י אירוע המספרה. הקלט יהיה אחד משלושה תווים:

H - מופע של לקוח חדש במספרה. הלקוח יצטרף לתור הממתינים לשרות.
Y - יציאה של לקוח מן התור לאחר סיום השרות.
D - הדפסה של רשימת הממתינים במספרה בנקודת זמן כלשהי.

התכנית לאירוע המספרה:

```
PROGRAM  MISPARA (INPUT, OUTPUT);
TYPE
     TAV10 = PACKED ARRAY [1..10] OF CHAR;
     MATZBIA = ^TZOMET;
     TZOMET = RECORD
                SHEM: TAV10;
                HABA: MATZBIA;
                KODEM: MATZBIA
              END;
VAR
  P, ROSH, SOF: MATZBIA;
  KOD: CHAR;
  X: TAV10;

PROCEDURE  KRA_SHEM (VAR NAME : TAV10);
BEGIN
  WRITELN ('כתוב את השם של הלקוח שהגיע');
  READ (NAME);
  READLN
END;
```

```
PROCEDURE  HADPES;
BEGIN
  WRITELN;
  WRITELN ('^^^תור');
  P:= ROSH;
  WHILE P <> NIL  DO
    BEGIN
      WRITELN (P^.SHEM);
      P:= P^.HABA
    END
END;

PROCEDURE  MOFA (NAME : TAV10);
BEGIN
  NEW (P);
  P^.SHEM:= NAME;
  P^.HABA:= NIL;
  P^.KODEM:= SOF;
  IF SOF = NIL THEN
    ROSH:= P
  ELSE
    SOF^.HABA:= P;
  SOF:= P
END;

PROCEDURE  YETZIA;
BEGIN
  IF ROSH <> NIL THEN
   BEGIN
     P:= ROSH;
     ROSH:= ROSH^.HABA;
     IF ROSH <> NIL THEN
       ROSH^.KODEM:= NIL
     ELSE
       SOF:= NIL;
     DISPOSE (P)
   END
END;

 BEGIN              (* התכנית הראשית *)
   ROSH:= NIL;
   SOF:= NIL;
   WRITELN ('^^^^^^מופע לסמן "M" הדפס ');
   WRITELN ('^^^^^יציאה לסמן "Y"       ');
   WRITELN ('^^^^^הדפסה לבקש "H"       ');
   READLN (KOD);
   WHILE NOT EOF DO
      CASE KOD OF
         'M': BEGIN
                 KRA_SHEM (X);
```

```
(67)              MOFA (X)
(68)             END;
(69)         'Y': YETZIA;
(70)         'H': HADPES
(71)     END
(72) END.
```

הפלט של התכנית:

```
      ^^^^^^
הדפס 'M' לסמן מופע
     'Y' לסמן יציאה
     'H' לבקש הדפסה
M
כתוב את השם של הלקוח שהגיע
AVRAHAM
M
כתוב את השם של הלקוח שהגיע
BARUCH
Y
M
כתוב את השם של הלקוח שהגיע
FISHER
H
^^^
תור
BARUCH
FISHER
M
כתוב את השם של הלקוח שהגיע
GERSHON
M
כתוב את השם של הלקוח שהגיע
MOSHE
Y
H
^^^
תור
FISHER
GERSHON
MOSHE
```

הסבר:

הפרוצדורות KRA_SHEM ו-HADPES אינן שונות מפרוצדורות דומות עבור רשימה משורשרת. בהדפסה מתחילים מראש התור (ROSH) ומתקדמים עד לסופו, כאשר P^.HABA=NIL.

(31)-(42) הפרוצדורה MOFA: יוצרים איבר חדש (שורה 33) שרושמים בו את המופע החדש, שעכשיו נמצא בסוף התור (שורה 34). את המצביע לאחור של האיבר החדש מכוונים ל-NIL (35), כי האיבר החדש הוא תמיד האחרון בתור. את המצביע לפנים של האיבר החדש מכוונים אל האיבר שעד עכשיו היה האחרון (36). אם התור היה ריק עד למופע זה (SOF=NIL), צריך לשנות את הכתובת ב-ROSH מ-NIL ל-P (38). אם התור לא היה ריק, צריך לשנות את המצביע של

האיבר שהיה עד עכשיו האחרון בתור כדי שיצביע על האיבר של המופע החדש (40). בכל מקרה, הכתובת של האיבר החדש היא הערך החדש של SOF (41).

(43)-(55) הפרוצדורה YETZIA: כאשר מסתיים השרות של הלקוח, הוא יוצא מראש התור. לכן, אם יש מישהו בתור (כלומר, ROSH<>NIL בשורה (45)), עוברים לראש התור ומורידים את האיבר שנמצא שם (שורות (47), (48), (53)). אם התור החדש איננו ריק, צריך לשנות ל-NIL את המצביע KODEM של האיבר שעד עכשיו היה שני בתור וזה יסמן שהוא עכשיו הראשון בתור (49)-(50). אם התור החדש ריק, אין צורך לעדכן את המצביעים של צומת כלשהו, אך צריך לעדכן את המצביע לסוף התור (SOF) שיהיה שוה ל-NIL (52).

(56)-(72) התכנית הראשית מורכבת מלולאה אחת שמקבלת את קוד הפעולה (מופע, יציאה והדפסה) וקוראת שם לקוח, אם יש מופע. התכנית מסתיימת כאשר מגיעים לסוף הקלט.

17.5 עצים

מבנה עץ (Tree) אפשר להחזיק כרשימה משורשרת עם שני מצביעים לשני מקומות במערך, כלומר לשני איברים סטטיים. אפשר גם להחזיק ברשומה מצביעים לשני איברים דינמיים. נכתוב תכנית כדי לבנות ולהדפיס עץ.

```
PROGRAM  TREE (INPUT, OUTPUT);
TYPE
   MATZBIA = ^ETZ;
   ETZ = RECORD
            NATUN: SUG_NATUN;          (* מחליפים SUG_NATUN *)
            SMOL: MATZBIA;             (* בסוג אמיתי *)
            YAMIN: MATZBIA
         END;
VAR
     SHORESH: MATZBIA;
     KELET: SUG_NATUN;

PROCEDURE  BENIYA (VAR TZOMET: MATZBIA;
                   KELET_CHADASH: SUG_NATUN);
                               (* פרוצדורה זו מוסיפה צומת לעץ *)
    BEGIN
      IF TZOMET = NIL  THEN
        BEGIN                          (* נוצר צומת חדש *)
          NEW (TZOMET);
          TZOMET^.SMOL:= NIL;
          TZOMET^.YAMIN:= NIL;
          TZOMET^.NATUN:= KELET_CHADASH
        END
      ELSE
        BEGIN
          IF KELET_CHADASH < TZOMET^.NATUN THEN
```

```
               BENIYA (TZOMETˆ.SMOL, KELET_CHADASH);
            IF KELET_CHADASH >  TZOMETˆ.NATUN THEN
               BENIYA (TZOMETˆ.YAMIN, KELET_CHADASH)
          END
      END;                                   (* BENIYA *)

      PROCEDURE  HADPASA (TZOMET: MATZBIA);
      BEGIN            (* הדפס את הנתונים לפי סדר מיון עולה *)
        IF TZOMET <> NIL  THEN       (* בדוק אם צריכים להפסיק *)
          BEGIN
            HADPASA (TZOMETˆ.SMOL);
            WRITE (TZOMET.NATUN);
            HADPASA (TZOMETˆ.YAMIN)
          END                                  (* הדפסת צמתים *)
      END;                   (*  HADPASA  *)

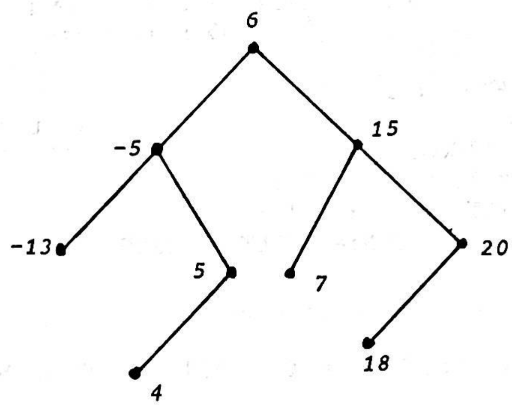
    BEGIN                                      (*  תכנית ראשית *)
      SHORESH:= NIL;
      WRITELN ('  הכנס ערכים');
      READ (KELET);
      BENIYA (SHORESH, KELET);
      WHILE NOT EOF DO
        BEGIN                                    (* בניית עץ *)
            BENIYA (SHORESH, KELET);
            READ (KELET)
        END;
      WRITELN;                                   (* דלג שורה *)
      WRITELN  ('התוכן של העץ מופיע למטה');
      HADPASA (SHORESH)
    END.
```

לדוגמא, נקרא ערכים ונבנה עץ.

לקרוא משמאל לימין:

6   -5   15   5   -13   20   7   -5   15   18   4

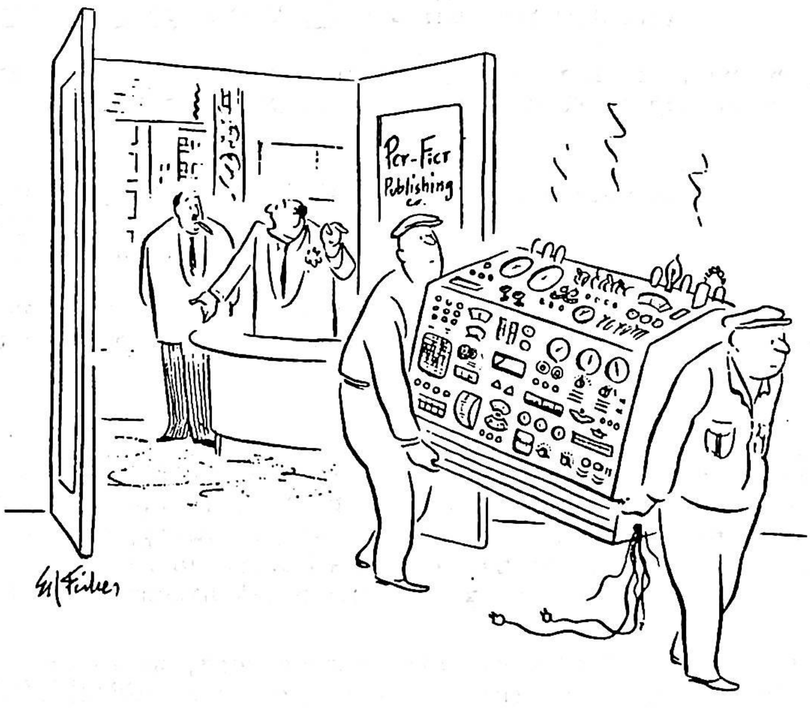
העץ שנבנה על פי התכנית:



17.6 תרגילים

(1) שנה את הפרוצדורה HASARA כדי שתבדוק אם המחסנית ריקה.

(2) תכניות 1 עד 15 שבפרק 16, מבנה נתונים, מתאימות גם לפרק זה. יש להשתמש רק במבנים דינמיים.



אנחנו אבודים. רק הוא הבין מה הולך כאן.

נספח א'

המחשב ושיטות ספירה

1. מערכת המחשב

1.1 יחידת העיבוד המרכזית - יע"מ

אפשר לסווג את הרכיבים של מחשב בשתי קבוצות:

- יחידת עיבוד מרכזית - יע"מ, או (Central Processing Unit- CPU).
- יחידות קלט/פלט (Input/Output Units - I/O Units).

קבוצת רכיבי הקלט/פלט מורכבת מיחידות הקלט ויחידות הפלט, כמו מדפסת, מצג (מסוף), דיסק, תקליטון ועוד. קבוצת רכיבי ה-CPU כוללת את שלוש היחידות הבאות:

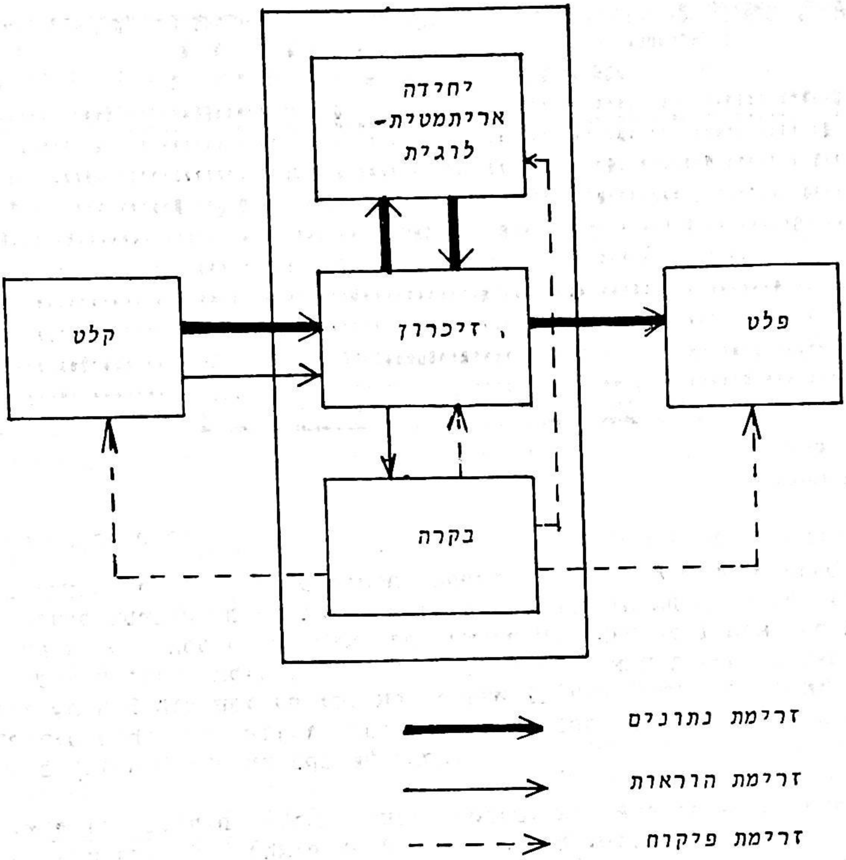
- יחידה אריתמטית-לוגית (Arithmetic-Logic Unit - ALU).
- זיכרון (Memory).
- בקרה (Control).

הקשר בין הרכיבים האלה נראה בציור הבא, כאשר חץ רגיל מסמן זרימת הוראות, חץ עבה מסמן זרימת נתונים וחץ מקווקו מסמן זרימת אותות בקרה ופיקוח.

החלוקה של היחידות הבסיסיות במחשב מוסכמת על הכל, אך יש חילוקי דעות לגבי המינוח וההקבצה שלהן. יש שמחילים את השם יע"מ, או CPU, לא על שלוש היחידות שהוזכרו קודם, אלא רק על היחידה האריתמטית-לוגית, כאשר הזיכרון ויחידת הבקרה נפרדים מן ה-CPU. אחרים כוללים רק שתי יחידות, את יחידת הבקרה והיחידה האריתמטית-לוגית במושג של היע"מ.

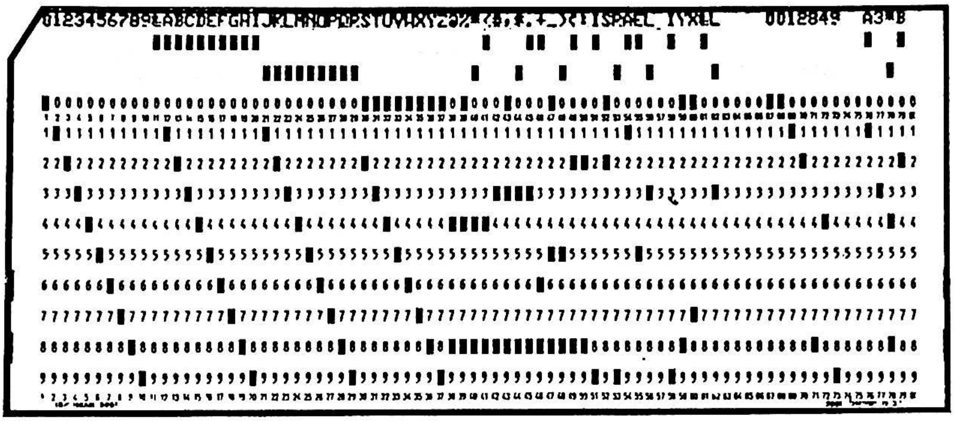
1.2 הכרטיס המנוקב

מחזור החיים של נתונים שמעובדים ע"י מחשב מתחיל מחוץ למחשב ומסתיים מחוץ לו. כדי להעביר את הנתונים לתוך המחשב נעזרים ביחידות קלט.



עד סוף שנות השבעים, השיטה המקובלת ביותר לקליטת נתונים ע"י מחשב היתה מורכבת משני צעדים. בתחילה היו מקישים את הנתונים בכרטיסים במכונה לניקוב כרטיסים. המכונה ניקבה מ-1 ועד 3 חורים בגוף הכרטיס כדי שישמשו קוד, אשר המחשב יוכל לקרוא אותו ביחידת הקלט. בנוסף, המכונה היתה מדפיסה את התווים על השפה העליונה של כרטיס כדי שאנשים יבינו את תוכנו. כל תו נוקב והודפס בעמודה אחת מתוך 80 העמודות של הכרטיס.

השלב השני היה הזנה של הכרטיסים האלה למכונה אחרת, קוראת כרטיסים. מכונה זו היתה מזהה את הנקבים, מפענחת את הקוד ומעבירה אל המחשב עותק של הנתונים מהכרטיס שנקרא.



עיין בתמונת הכרטיס:

המספרים בשפה הימנית של הכרטיס הם מספרי השורות. המלבנים המאורכים למטה מכל תו מודפס מסמנים את מקומם של החורים אשר מהווים את הקוד של אותו תו. למשל, הקוד של הספרה 0 הוא נקב בשורה 0, הקוד של 1 הוא נקב בשורה 1 וכדומה לשאר הספרות. הקוד עבור A הוא שני נקבים: אחד בשורה 12 ואחד בשורה 1. הקוד עבור B הוא שני נקבים: אחד בשורה 12 ואחד בשורה 2. לשאר האותיות שיטת הקודים דומה, אלא שזוגות הנקבים כוללים נקב בשורה 11 יחד עם נקב של ספרה ונקב בשורה 0 יחד עם נקב של ספרה.

הקוד <u>לאותיות בעברית</u> מורכב משני נקבים, אשר חופפים את הניקוב לתווים באנגלית, פרט לאות "א" (האות ה-27), אשר חורגת ומנוקבת כנקב נפרד בשורה 12. הקוד לסימנים מיוחדים הוא בן 3 נקבים כמו למשל הקוד עבור סימן החיבור (+) אשר כולל 3 נקבים בשורות 12, 6, 8.

מספר העמודות בכרטיס הוא 80 ולכן המספר המירבי של תווים שאפשר היה להכיל בכרטיס אחד היה 80 תווים. מן הראוי לציין שהיו בשימוש גם כרטיסים בעלי מבנה שונה.

<u>1.3 עבודה במסוף</u>

בכלים אשר נמצאים ברשותנו היום ניתן להזין נתונים למחשב בשלב אחד. במקום להשתמש במכונת הניקוב, משתמשים במסוף לשם הזנה של הקלט, אשר מועבר באופן ישיר אל המחשב בלי התיווך של כרטיסים. בתקופת השימוש בכרטיסים, מכונת הניקוב לא היתה מחוברת אל המחשב. קוראת הכרטיסים בלבד היתה מחוברת אליו ועל כן רק היא נחשבה יחידת קלט.

המסוף, אשר משלב את התפקידים של מכונת הניקוב ושל קורא הכרטיסים, הפך להיות מכשיר הקלט הנפוץ ביותר. הנתונים עוברים מן המסוף אל זיכרון המחשב ללא התערבות ידנית נוספת. העברה זאת נעשית כתוצאה מהוראה של יחידת הבקרה,

אשר מפקחת על תהליך העיבוד מתחילתו עד סופו. לכן, בציור למעלה, יש חץ מקווקו שמקשר בין יחידת הבקרה לבין יחידת הקלט. אין זרימה של נתונים לעיבוד בין שתי יחידות אלה, אלא של אותות פיקוח שמורים ליחידת הקלט מה ואיך לעשות.

1.4 זיכרון המחשב

בזיכרון המחשב מחזיקים 3 סוגים של נתונים:

א. הוראות למחשב.
ב. נתונים.
ג. תוצאות ביניים ותוצאות סופיות.

סוגי נתונים א' ו-ב' מגיעים לזיכרון מן החוץ באמצעות מסוף או באמצעות יחידת קלט אחרת. הסוג השלישי נוצר כתוצאה מעיבוד של נתוני הקלט ע"י הוראות התכנית.

קבוצת הוראות אשר מהווה יחידת עיבוד אחת נקראת <u>תכנית</u> (Program). כדי לבצע את התכנית קוראים אותה לזיכרון המחשב מדיסק או תקליטון. בגמר הביצוע, תכנית אחרת תופשת את מקומה כדי להתחיל תהליך עיבוד חדש.

הזיכרון משמש לנתונים כיחידת המתנה. הם נמצאים שם אחרי שמעבירים אותם מיחידת הקלט ולפני שמעבירים אותם לעיבוד ולפלט.

ההוראות (המהוות תכניות) מועברות כסידרן ליחידת הבקרה, שבה מפענחים כל הוראה ומבררים איזו פעולה על המחשב לבצע. נתונים אינם מועברים ליחידת הבקרה. בהתאם להוראות התכנית הם מועברים ממקום למקום בזיכרון, או מן הזיכרון ל-ALU. זהו תהליך עיבוד הנתונים.

מכיון שהעברות אלו הן תחת הפיקוח של יחידת הבקרה, נרשם חץ מקווקו בשרטוט של היע"מ, בין יחידת הבקרה לבין הזיכרון. מעבירים נתונים ממקום למקום בזיכרון אם רוצים לשלב יחד נתונים חלקיים אשר נמצאים בו במקומות שונים בצורה לא רציפה, או לא בסדר הדרוש לשורת הפלט. מעבירים נתונים ל-ALU אם יש פעולות אריתמטיות או לוגיות (ז"א השוואות מסוגים שונים) שרוצים לבצע עליהם.

כאשר המחשב מסיים לבצע פעולות אלו, יחידת הבקרה מחזירה את התוצאות לשמירה בזיכרון. זהו החץ העבה הדו-כיווני בין הזיכרון וה-ALU. החץ המקווקו מיחידת הבקרה ל-ALU מסמן שיחידת הבקרה יוזמת העברת התוצאות חזרה לזיכרון בתום פעולת החישוב.

בסיום ביצוע התכנית, המחשב שוב פעם צריך להתקשר עם העולם החיצון והוא עושה זאת בעזרת יחידות הפלט. יחידות הפלט הנפוצות ביותר הן המדפסת, אשר מפיקה את הפלט בנייר והמסוף אשר מציג את הנתונים על גבי המסך. יחידות פלט חשובות הן יחידות ההחסנה של הנתונים, כמו הסרט המגנטי, הדיסק והתקליטון. ביחידות אלו מחסנים את תוצאות העיבוד לשם עיבוד נוסף במועד אחר ואז הן משמשות כיחידות קלט.

1.5 ייצוג תווים, מספרים, סיביות ובתים

כל הנתונים שהמחשב מעבד צריכים לשהות בשלב כלשהוא בזיכרון. הזיכרון בנוי מרכיבים חשמליים שיכולים להיות באחד משני המצבים, דלוק (ON) או כבוי (OFF) ובמונחים מתמטיים - 1 או 0.

כל רכיב נקרא סיבית (BIT, קיצור של Binary DigIT). מכיון שדרושות סיביות רבות בכדי לייצג תווים שונים, מקבצים אותן יחד לקבוצות של 8. צירוף של 8 סיביות נקרא בית (Byte). הזיכרון של מחשב גדול מכיל מיליוני בתים, למשל, 32 מיליון. שם מקוצר לאלף בתים הוא קילו-בית (K-BYTE, KILOBYTE) ושם מקוצר למיליון בתים הוא מגה-בית (Megabyte).

מה אפשר להחזיק בבית אחד? כאמור, מחרוזת של אותות חשמליים אשר מורכבת מיחידות של 0 ושל 1. כמה מחרוזות שונות כאלו יש? היות והבית מכיל 8 סיביות, שכל אחד מהם יכול להיות באחד משני מצבים, אפשר להחזיק בו 256 צירופים שונים ($256 = 2^8 = 2*2*2*2*2*2*2*2$), או מחרוזות שונות של סיביות.

מה יכולים לייצג בעזרת 256 המחרוזות האפשריות השונות של סיביות?

א. ספרות (0-9) - דרושות 10 מחרוזות כדי לייצג את כל הספרות.
ב. אותיות לטיניות גדולות וקטנות - דרושות 52 (2*26) מחרוזות שונות.
ג. סימנים מיוחדים - למשל +, -, *,), (, ועוד.
ד. אותיות בעברית - דרושות עוד 27 מחרוזות.
ה. תווי בקרה שונים - דרושות מחרוזות בהתאם לצורך. תווי הבקרה דרושים בעיקר לניהול תקשורת בין המחשב לבין היחידות השונות המשמשות לעיבוד מקוון (Online Processing).

צירוף של 8 סיביות יכול לייצג ספרה, אות, או סימן כלשהו ונכנה אותו בשם תו (Character). מספר המחרוזות שאפשר לבנות בבית של 8 סיביות הוא 256. מספר זה גבוה ממספר התווים שנצטרך לייצג.

איזו מחרוזת מסוימת של סיביות "אפס" וסיביות "אחד" ייצג כל תו? קיימות כמה שיטות קידוד לתווים, כפי שנתאר בהמשך. בשלב זה די לדעת שמחרוזת של 8 סיביות יכולה לייצג כל תו שנרצה. לדוגמא,

נניח שהקוד עבור הספרה 4 הוא: 00 110 100

והקוד עבור הספרה 0 הוא: 00 110 000

לפיכך, הייצוג הפנימי של המספר 4000 יהיה:

|——4——| |——0——| |——0——| |——0——|
00 110 100   00 110 000   00 110 000   00 110 000

לייצוג זה דרושים 4 בתים, שהם 32 סיביות.

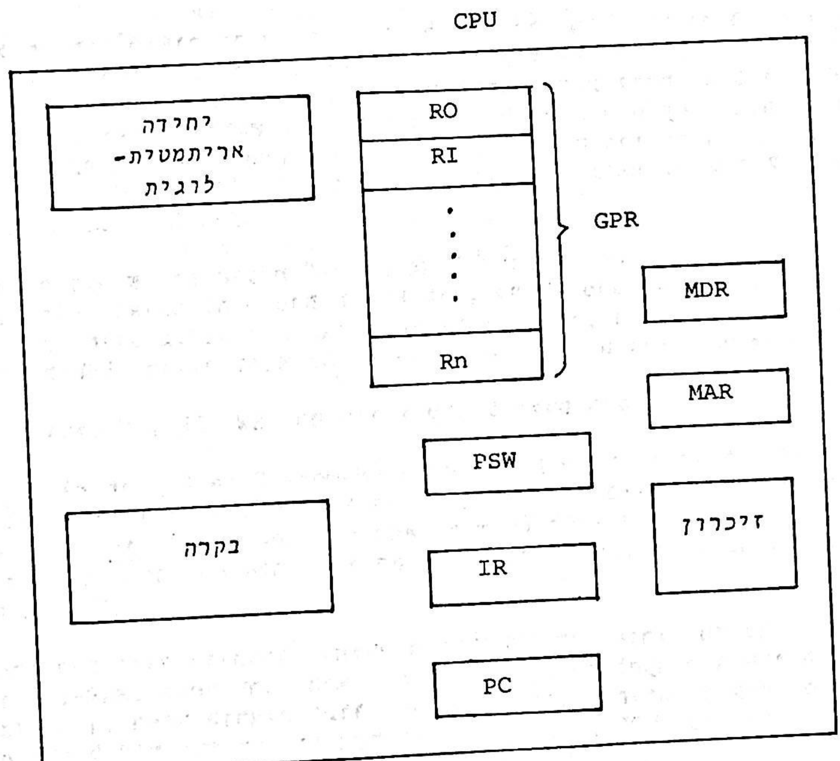
כמו שאפשר לשלב יחד 8 סיביות כדי לקבל בית אחד, נוכל לשלב יחד מספר בתים כדי לקבל מלה (Word). במחשב PDP-11 למשל, המלה מורכבת משני בתים (16 סיביות), כאשר במחשבים יותר גדולים כדוגמת מחשבי יבמ בארכיטקטורת 370, או

מחשבי VAX, המלה מורכבת מ-4 בתים (32 סיביות). חשוב להכיר את המושג מלה מכיון שהגודל של האוגרים (Registers), שבהם מתבצעות פעולות אריתמטיות ופענוח הוראות הוא בדרך כלל כגודל של המלה של המחשב. נובע מזה שהערך המספרי הגבוה ביותר והנמוך ביותר וגם ההוראה הטיפוסית, נקבעים לפי אורך המלה.

1.6 אוגרים

אוגר (Register) הוא אביזר ביע"מ שמסוגל לקלוט נתונים, להחזיק אותם, ולהעביר אותם בין שלבי עיבוד שונים בתוך היע"מ. הוא מיועד בעיקר להחזקה זמנית של נתונים לביצוע פעולות חישוב ופעולות לוגיות. מכיון שהוא מיועד לעבודה עם נתונים מספריים, גודלו צריך להיות כגודל מלה באותו מחשב.

התיאור של היע"מ והאוגרים העיקריים הניתן בתרשים הינו סכימתי, לשם הסבר מושגים ולא לשם לימוד של מחשב מסוים.



בזיכרון של המחשב מחזיקים גם נתונים וגם הוראות (או פקודות). הוראה טיפוסית יכולה להראות כך:

כתובת שנייה	כתובת ראשונה	קוד הפעולה

הקוד מציין את סוג הפעולה, למשל חיבור, חיסור, העברה ממקום למקום או השוואה. כתובת בזיכרון היא המספר הסידורי של תא מסוים (מספר הבית או מספר המלה) בזיכרון. הכתובת הראשונה בהוראה מציינת את מיקומו של הערך שנבצע בו את הפעולה המבוקשת (האופרנד הראשון). הכתובת השנייה מציינת את מקומו של האופרנד השני, אם מדובר על פעולה הזקוקה לשני אופרנדים. התוצאה של הפעולה תאוחסן בכתובת השנייה ותחליף את תוכנה הקודם.

ההוראות מוחסנות בזיכרון באופן סודר, ז"א אחת אחרי חברתה ויוצרות תכנית. בזמן ביצוע התכנית, יחידת הבקרה מעבירה את הכתובת של ההוראה הראשונה לאוגר שנקרא מונה התכנית (PC – Program Counter). יחידת הבקרה בודקת את האוגר PC ומעבירה את הפקודה שנמצאת בכתובת שמופיעה ב-PC אל אוגר ההוראות (IR -Instruction Register). בכל פעם שהוראה מועברת, התוכן של PC גדל ב-2 (אם אורך ההוראה הוא שני בתים) ולכן הוא מצביע תמיד על הפקודה הבאה שבתור לביצוע. לאחר פיענוח וביצוע ההוראה הנוכחית, יחידת הבקרה תביא עוד הוראה מן הכתובת שב-PC.

בנוסף על פקודות חישוב והעברות ממקום למקום בזיכרון, יש גם פקודות דילוג, או הסתעפות (Branching Instructions) שמנתבות את הביצוע אל הוראה שנמצאת במקום אחר, ולא אל ההוראה העוקבת. ביצוע הוראה כזאת גורם לשינוי התוכן של PC לכתובת של ההוראה שמבקשים להסתעף אליה. לפעמים ההסתעפות מותנית בתוצאה של פעולה קודמת. למשל, נסתעף אם התוצאה של הפעולה הקודמת היתה חיובית או שלילית, או שגרמה לגלישה. מצב של גלישה קורה כאשר מגיעים לתוצאה אשר גדולה מהקיבולת של האוגר.

בסיום כל פעולה שומרת היע"מ "תמונת מצב" באוגר מיוחד הקרוי PSW (Process Status Word). באוגר זה שומרת היע"מ קוד, שערכו מסמל אם התוצאה של הפעולה הקודמת היתה חיובית, שלילית, אפס, או גלישה. לכן, בעת בדיקת קיום תנאי יש לפנות אל ה-PSW. פנייה זו אינה ישירה, אלא באמצעות פקודה מתאימה.

שלושת האוגרים PC, IR, PSW הם אוגרי עזר ליחידת הבקרה.

האוגר MDR (Memory Data Register) משמש כתחנת מעבר לנתונים בין יחידות הקלט/פלט, או אוגרי היע"מ לבין הזיכרון. הכתובת בזיכרון לקריאה או כתיבה מועברת לאוגר MAR (Memory Address Register) לפני שפעולות אלו מתבצעות. מפאת התפקיד אשר ממלאים שני האוגרים MAR ו-MDR, אפשר לראות אותם כחלק מיחידת הזיכרון.

כאשר מחסנים נתון בזיכרון, יחידת הבקרה מעבירה אותו תחילה ל-MDR ואת כתובתו - ל-MAR. לאחר מכן, היא משגרת אות שגורם להעברת הנתון הזה לכתובת היעד בזיכרון. כאשר קוראים נתון מן הזיכרון, יחידת הבקרה מעבירה את הכתובת שלו ל-MAR ומשגרת אות אשר גורם להעברתו ל-MDR. משם הוא מועבר ל-IR אם הוא הוראה, ואם הוא נתון לעיבוד, הוא מועבר ל-ALU, למקום אחר בזיכרון, או ליחידת פלט.

בכל יע"מ יש גם אוגרים כלליים (General Purpose Registers - GPR). הם מיועדים לביצוע פעולות אריתמטיות ופעולות לוגיות במספרים בינריים ובמספרים המוצגים בשיטת נקודה צפה (ראה הסבר בהמשך).
המתכנת יכול להשתמש באוגרים הכלליים באופן ישיר כאשר הוא מתכנת בשפת סף (Assembler). שפות תכנות עיליות (High Level Languages), כדוגמת פסקל משתמשות באוגרים אלו גם כן, מבלי שהמתכנת יציין זאת במפורש. אפשר להסתכל על אוגרים אלה כהשלמה לאוגרים הפנימיים של ה-ALU. במחשבי יבמ בארכיטקטורה 370 יש 16 אוגרים כלליים ו-4 אוגרים לשיטת הנקודה הצפה, במחשב PDP-11 יש 6 אוגרים אשר פנויים לשימוש המתכנת וב-VAX יש 12 אוגרים.

מן הראוי לציין שהמבנה הארכיטקטוני של היע"מ שונה במחשבים שונים. כך למשל, במחשבי יבמ ישנם אוגרים נפרדים כדי לבצע פעולות חישוב ופעולות לוגיות בתווים ובמספרים עשרוניים, בנוסף לאוגרים לפעולות בשיטת הנקודה הצפה.

2. מספרים בינריים ובסיסי מספרים

מכל שנאמר אודות ייצוג ספרות בזיכרון של המחשב, נובע שמחשב שבו המלה היא בת 32 סיביות יכול להחזיק מספר שגודלו המקסימלי הוא 4 ספרות (4 בתים) ומחשב שבו המלה היא בת 16 סיביות יוכל להחזיק מספר שגודלו עד 2 ספרות.

הקורא אולי תמה על הערך העשרוני הקטן של המספר המקסימלי, שלכאורה המחשב יכול להחזיק במלה. ערך זה הוא 99, אם אורך המלה הוא 2 בתים, או 9999 אם אורך המלה הוא 4 בתים. התשובה לשאלה זו היא, שיש צורה יותר יעילה להחזיק מספרים, אשר מבוססת על מקורה של המלה סיבית (BIT), דהיינו ספרה בינרית (Binary digIT). הכוונה היא שאפשר לראות כל סדרה של ספרות 0 ו-1 כמספר בבסיס 2. למשל, אפשר לראות את המספר 101 כאילו הוא שווה ל:

$$1*2^2 + 0*2^1 + 1*2^0 = 5$$

אפשר להציג בצורה כללית כל מספר לא שלילי N כמספר בינרי:

$$n_{m-1}\ \ n_{m-2} \cdots n_0$$

ערכו של המספר יהיה:

$$N=\sum_{i=0}^{m-1} n_i 2^i$$

כאשר כל n_i יהיה שווה או ל-0 או ל-1.

להלן טבלה של 8 המספרים הבינריים שאפשר להציג ב-3 סיביות:

N	n_2 n_1 n_0
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

בצורה כללית, אפשר להציג מספרים בכל בסיס $b \geq 2$ כסדרה של m ספרות:

$$n_{m-1} \quad n_{m-2} \quad \dots \quad n_1 \quad n_0$$

כאשר: א. $0 < n_i < b-1$

ב. $N < b^m$

ג. $N = \sum_{i=1}^{m-1} n_i b^i$

ערכים מקובלים בעבור הבסיס b הם: 10 (דצימלי - Decimal), 2 (בינרי - Binary), 8 (אוקטלי - Octal) ו-16 (הקסדצימלי - Hexadecimal).

מספר הספרות השונות בשיטת ספירה מסוימת הוא כגודל הבסיס. בשיטה הבינרית יש שתי ספרות (0,1), בעשרונית יש 10 (0-9) ובאוקטלית ובהקסדצימלית יהיו 8 ו-16 ספרות, בהתאמה. בטבלה מוצגות הספרות האפשריות בכל אחד מ-4 הבסיסים:

שם	בסיס	ספרות
בינרי	2	0,1
אוקטלי	8	0,1,2,3,4,5,6,7
הקסדצימלי	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
דצימלי	10	0,1,2,3,4,5,6,7,8,9

המספר הגדול ביותר שנוכל להחזיק ב-m סיביות הוא b^m-1. למשל, אם בשיטה הבינרית m=3, המספר הגדול ביותר שאפשר להחזיק הוא 7 ($111=2^3-1=7$).

בטבלה למטה מציגים את המספר העשרוני 37 בבסיסים שונים:

בסיס	ייצוג	פירוט החישוב
בינרי	100101	$1*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0$
אוקטלי	45	$4*8^1 + 5*8^0$
הקסדצימלי	25	$2*16^1 + 5*16^0$

מתוך הטבלה אפשר ללמוד, שאם נקבץ את הספרות הבינריות בקבוצות של 3, ונחליף כל שלישיה בספרה אוקטלית, נקבל ייצוג אוקטלי. המספר הבינרי בטבלה הוא 100101. נקבץ אה הספרות הבינריות לשלישיות ונקבל:

101 100 ===> 45 אוקטלי
5 4

אם נקבץ את הספרות הבינריות ברביעיות ונחליף כל רביעיה בספרה הקסדצימלית, נקבל ייצוג הקסדצימלי, כפי שמוצג למטה:

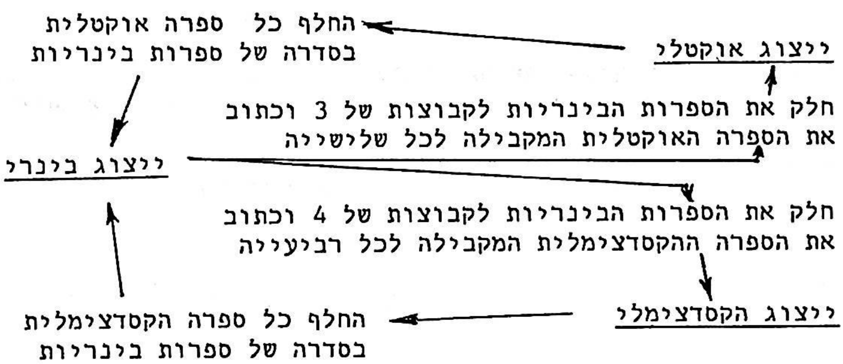
0101 10 ===> 25 הקסדצימלי
5 2

נוכל גם לפעול בכיוון הפוך ולהתחיל עם המספר 45 אוקטלי, לכתוב את ה-5 ואת ה-4 כסדרה של ספרות בינריות (101, 100, בהתאמה) ולשלב אותם יחד כדי לקבל את המספר המקורי (45) בייצוג בינרי (101 100).

2.1 מעבר בין שיטות ספירה

על פי ההסבר שניתן אפשר לשרטט תרשים למעבר בין שיטת ספירה אחת על בסיס שהוא חזקה של 2 לכל שיטה אחרת על בסיס כזה.

מעבר בין שיטות ספירה: בינרית, אוקטלית והקסדצימלית



כדי לעבור מייצוג אוקטלי לייצוג הקסדצימלי צריך לעבור תחילה דרך הייצוג הבינרי. לדוגמא, נציג את המספר הדצימלי 112 בשיטה האוקטלית:

מספר אוקטלי	0 | 6 | 1
העברה לייצוג בינרי	0 0 0 1 1 1 0 0 0
הקבצה לרביעיות כדי לעבור לייצוג הקסדצימלי	0 | 7 |

ניתן להסיק מן האמור לעיל:

א. אפשר לתאר את התוכן של מלת המחשב כמספר הקסדצימלי, אוקטלי או בינרי. בכל מקרה, הרכיבים בעצמם הם סיביות והשאלה היחידה היא אם מסתכלים על כל סיבית בנפרד או מקבצים אותן בקבוצות של 3 סיביות כדי לקבל מספר אוקטלי, או בקבוצות של 4 סיביות כדי לקבל מספר הקסדצימלי.

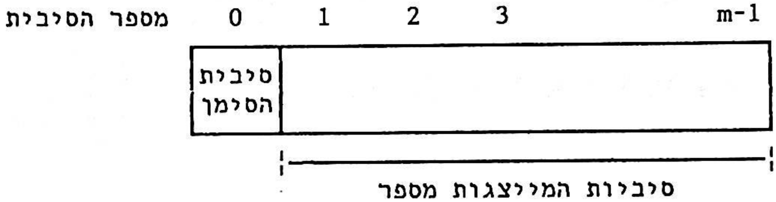
ב. נוכל להסתכל על סדרת הסיביות בכל בית כמספר בינרי, או כקוד שמטרתו לייצג תו כלשהו.

2.2 מספרים שליליים

באריתמטיקה מייצגים מספרים שליליים ע"י שימוש בסימן מינוס (-). סימן זה אינו מתאים לשימוש במחשב, שבו קיימות רק הספרות 0 ו-1, אשר מייצגות את שני המצבים של האותות החשמליים.

משתמשים בשתי ספרות אלו גם לייצוג הסימן: "פלוס" (+) ע"י הספרה 0, ו"מינוס" (-) ע"י הספרה 1. עתה יש לקבוע כללים איך להבדיל בין 0 המסמל את הספרה "אפס" לבין 0 המסמל "פלוס" ובין 1 המסמל את הספרה "אחד" לבין 1 המסמל "מינוס". התשובה היא שהמיקום של ה-0 או ה-1 במלה הוא הקובע.

נתונה מלת מחשב בת m סיביות:



הסימן נקבע ע"י הערך הבינרי של סיבית הסימן. הערך של המספר נקבע ע"י מחרוזת הסיביות שמופיעה בשאר המקומות של המלה, מימין לסיבית הסימן.

הערך של המספר נקבע על פי שיטת הייצוג שלו. קיימות שלוש שיטות שונות לייצוג של מספרים שלמים ושיטה אחת כדי לייצג מספרים בשברים.

3. שיטות לייצוג של מספרים שלמים

3.1 שיטת סימן וערך

שיטת סימן וערך (Sign and Magnitude) היא הפשוטה ביותר. סיבית הסימן היא 0 למספר חיובי ו-1 למספר שלילי. המספר עצמו מיוצג כמספר בינרי ע"י סדרת הסיביות שמימין לסימן.

	ערך	סימן
למשל, המספר 5- ייראה כך:	0000...101	1
המספר 5+ יוצג כך:	0000...101	0

שיטה זו פשוטה מאד ועל כן נסביר את חסרונותיה כדי להצדיק את השימוש בשיטת ייצוג אחרת. החסרונות של שיטה זו הם שלושה:

א. המספר 0 מיוצג בשתי דרכים: 1 ¦ 00.....0
(בשני סימנים שונים) 0 ¦ 00.....0

ב. כאשר מחברים שני מספרים, צריך לבדוק את סיבית הסימן של כל אחד מהם. אם הסימנים זהים, צריך לחבר אותם ואם הסימנים שלהם שונים, צריך לחסר ולא לחבר אותם. בשיטות אחרות, בדיקה זו אינה נחוצה.

ג. אם מרחיבים את המקום שנתפס ע"י מספר שלילי, למשל מ-16 סיביות ל-32 סיביות, צריך לאפס את סיבית הסימן במלה הימנית ול"הדליק" את סיבית הסימן במלה השמאלית, כמו שנראה בדוגמא:

מצב קודם: 16 סיביות — 0 ... 15: |1. ....|

מצב חדש: 32 סיביות — 0 ... 15: |0................| 0 ... 15: |1..............|

3.2 שיטת המשלים לשניים

בייצוג המשלים לשניים (Two's Complement) סיבית הסימן נמצאת בסיבית מספר m משמאל. היא שווה ל-0 עבור מספר חיובי ושווה ל-1 עבור מספר שלילי, כמו בשיטת סימן וערך. בשיטה זו, סדרת הסיביות בצד ימין היא ייצוג בינרי של המספר X רק אם X חיובי. אם X שלילי, סדרת הסיביות בצד ימין היא המשלים לשניים של X. המשלים לשניים של X עם m-1 סיביות מוגדר כהפרש בין 2^{m-1} לבין הערך המוחלט של X. הערך 2^{m-1} חייב להיות גדול או שווה לערך המוחלט של X.

אבל איך מוצאים את המשלים לשניים של מספר X?

יהי Z סדרת הסיביות ששווה לערך המוחלט של X. המשלים לשניים של X יהיה 2^{m-1}-Z. ביטוי זה אפשר לכתוב בצורה אחרת, 2^{m-1}-1-Z+1. דרך זו נוחה יותר לבניית אלגוריתם לחישוב המשלים של המספר X בשלושה שלבים:

א. חישוב 2^{m-1}-1
ב. חישוב 2^{m-1}-1-Z
ג. חישוב [(2^{m-1}-1)-Z]+1

דוגמא: נתון X=-53. המספר Z הוא הייצוג של |X| כמספר בינרי שערכו: 0110101. נניח ש m=8 ואז m-1=7. עכשיו נבצע צעדים אלה:

א. חישוב 2^{m-1}-1:
המספר הבינרי הגדול ביותר שאפשר להחזיק ב-3 סיביות הוא 111=2^3-1. בצורה כללית, המספר הבינרי הגדול ביותר שאפשר להחזיק ע"י m-1 סיביות כולל m-1 פעמים את הסיפרה 1. בדוגמא זו נקבל: 1111111=2^{8-1}-1.

ב. חישוב (2^{m-1}-1)-Z:

	1111111
הערך המוחלט של המספר Z	0110101
בסיום צעד ב' נקבל	1001010

לאחר צעד ב', מקבלים מספר שהוא היפוך הסיביות של המספר הנתון. ז"א, כל סיבית שהיתה 0 הפכה ל-1 וכל סיבית שהיתה 1 הפכה ל-0. נוכל עתה לשלב יחד צעדים א' ו-ב' לצעד אחד, שהוא הפיכת כל סיבית בייצוג הבינרי של הערך המוחלט של המספר המקורי. התוצאה של החלפות אלו נקראת המשלים לאחד (One's Complement) של מספר בינרי.

המספר הנתון	0110101
המשלים לאחד	1001010

ג. חישוב [(2^{m-1}-1)-Z]+1:
היות וחישבנו בשלב ב' את 2^{m-1}-1+Z, צריכים עכשיו רק להוסיף 1 לתוצאה זו ולקבל את הערך המבוקש.

המספר שמקבלים הוא המשלים לשניים (Two's Complement):

המשלים לאחד	1001010
הוספה של 1	+ 1
המשלים לשניים	1001011

ההוספה של 1 גרמה לכך שהספרה בצד ימין של המספר הפכה ל-1, כמו במספר המקורי. שאר הסיביות נשארו כמו שהיו בסוף שלב ב'.

מכאן נובעת שיטה מקוצרת להגיע למשלים של שניים של מספר נתון:

א. מעתיקים את הייצוג הבינרי של הערך המוחלט של המספר המקורי. מתחילים מימין עד שמגיעים לספרה הראשונה שערכה 1.
ב. לאחר שמזהים ספרה זו, "הופכים" כל סיבית משמאל לה.

נמצא את המשלים ל-2 של המספר הבינרי 100 0110 בדרך זו:

- נעתיק: 100
- נהפוך: 0110
- ונקבל: 1001 100

המשלים ל-2 של המשלים ל-2 של X שווה ל-Z, כאשר Z הוא הייצוג הבינרי של הערך המוחלט של המספר המקורי X. הסיבה לכך היא העובדה שהמשלים של X הוא $2^{m-1}-Z$, והמשלים ל-2 של אותו משלים ל-2 הוא $2^{m-1}-(2^{m-1}-Z)$ ששווה ל-Z.

הייצוג של המספר השלילי X כולל "1" בסיבית הסימן ו- $2^{m-1}-|X|$ ב-(m-1) הסיביות שמימין לו. ניתן לקצר תהליך זה ולחשב את הייצוג המלא בצעד אחד, כלומר: $2^m-|X|$, שהוא המשלים לשניים ב-m סיביות של המספר המקורי X.

שיטת ייצוג זו היא לכאורה מסובכת יותר מן השיטה הקודמת, אך נסקור את התכונות שלה ונעמוד על חשיבותה.

בשיטה זו ניתן לחבר שני מספרים מבלי להתחשב בסימנים שלהם ותמיד התוצאה תהיה נכונה. למשל, נניח שמחברים X ו-Y כאשר $X<0$, $Y>0$ ו- $|X|<Y$. הייצוג הפנימי של המספרים יהיה:

X שלילי

1	$2^{m-1}-\|X\|$

Y חיובי

0	Y

נשוא=1 ← נשוא=1

הסכום

0	$Y-\|X\|$

במקרה זה מצפים לתוצאה חיובית שהיא ההפרש בין Y לבין X, כלומר $Y-|X|$. נראה בהמשך איך מגיעים לתוצאה זו.

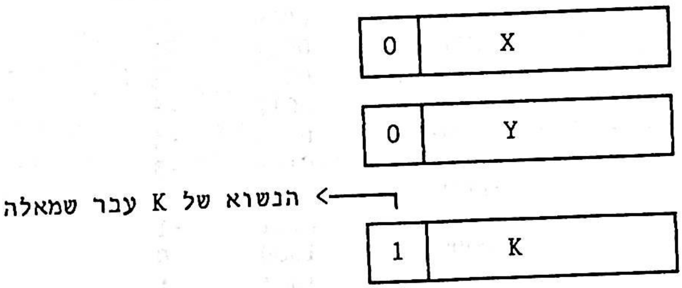
$$2^{m-1}-|X| + Y = 2^{m-1} + (Y-|X|)$$

המספר השלילי   המספר החיובי

את המספר 2^{m-1} אי אפשר לכתוב ע"י m-1 סיביות ולכן הוא נישא לסיבית מספר m, כאשר הסכום $Y-|X|$ נשאר בצד הימני של המלה. הנשוא של 1 אל מחוץ לסיבית הסימן הוא תוצאה מחיבור של ה-1 שהופיע בייצוג של המספר השלילי X והנשוא שהועבר כתוצאה מחיבור של $2^{m-1}-X$ ושל Y. התוצאה היא מספר חיובי (0 בסיבית הסימן) שערכו הוא $Y-|X|$ (הערך של הסיביות מימין לסיבית הסימן).

בתרגיל זה היו שני נשואים, אחד לתוך סיבית הסימן ואחד מחוצה לה. הכלל הוא שאם מספר הנשואים הוא זוגי (0 או 2), אין שגיאה בחיבור. אך אם מספר הנשואים הוא אי-זוגי נוצרת גלישה (Overflow), ז"א מצב שבו מאבדים נתונים, או מצב שבו מקבלים תוצאה שגויה.

למשל, מחברים שני מספרים חיוביים X ו- Y אשר מקיימים $X+Y>2^{m-1}$. אפשר לכתוב ביטוי זה כך: $X+Y=2^{m-1}+K$, כאשר K>0. כאשר הסכום של X ו-Y גדול מהערך 2^{m-1} מקבלים נשוא לתוך סיבית הסימן. אבל היות ושתי סיביות הסימן היו "אפס", לא יהיה נשוא נוסף מחוץ לסיבית הסימן. מצב כזה הוא מצב של גלישה, כפי שמוצג בתרשים הבא:



נדגים עכשיו את שני המקרים שתוארו קודם, כאשר m=4, כלומר סיבית אחת מוקדשת לסימן ו-3 סיביות לייצוג הערך של המספר.

א.

סיבית הסימן		
0	1 1 0	6
1	0 1 1	-5
0	0 0 1	1

מחברים שני מספרים, האחד חיובי והשני שלילי. תוצאת החיבור היא 1. הספרה 1 נישאת אל סיבית הסימן וגם מחוצה לה ולכן לא היתה גלישה.

ב.

0	1 0 1	5
0	1 1 0	6
1	0 1 1	11

בדוגמא זו מחברים שני מספרים שסכומם גדול מ-7, שהוא הערך המקסימלי שאפשר לייצג ב-3 סיביות (24-1-1=7). הספרה 1 בסיבית הסימן של התוצאה אומרת שהסיביות בצד ימין (011) הן המשלים לשניים של Z, כאשר Z הוא הייצוג הבינרי של הערך המוחלט של התוצאה. Z שווה אם כן, למשלים לשניים של 011 שערכו 101, דהיינו 5. התוצאה שווה 5- וזו אינה תוצאה אפשרית לסכום של שני מספרים חיוביים. העובדה שיש רק נשוא אחד לתוך סיבית הסימן ואין נשוא ממנה מצביעה על גלישה, כמו שהוזכר קודם, ולכן התוצאה שגוייה.

3.3 ייצוג של עודף 2^{m-1}

נניח שנתון $N > -2^{m-1}$. אם מוסיפים 2^{m-1} לכל ערך ש-N יוכל לקבל, התוצאה לא תהיה שלילית אף פעם, מכיון שהערך המינימלי של N הוא -2^{m-1} והוספנו לו 2^{m-1}. הייצוג של עודף 2^{m-1} (שנקרא Excess 2^{m-1}) של מספר N הוא אם כן, הערך $N+2^{m-1}$.

המספר שייצוגו בשיטת עודף 2^{m-1} שווה ל-A הוא $A-2^{m-1}$. למשל, אם A=1000 בייצוג של עודף 2^{4-1}, המספר המיוצג N הוא:

(אלה הם מספרים בינריים) $N = 1000 - 2^{4-1} = 1000 - 1000 = 0$

ייצוג של עודף 2^{m-1} של מספר זהה למשלים לשניים של אותו מספר חוץ מסיבית הסימן, אשר מוצגת בהיפוך. השוואה זו מוצגת בטבלה עבור המקרה שבו m=4, ז"א כאשר 3 סיביות מוקדשות לייצוג ערך המספר.

דצימלי	עודף 2^{4-1}	משלים ל-2
-8	0000	1000
-7	0001	1001
-6	0010	1010
-5	0011	1011
-4	0100	1100
-3	0101	1101
-2	0110	1110
-1	0111	1111
0	1000	0000
1	1001	0001
2	1010	0010
3	1011	0011
4	1100	0100
5	1101	0101
6	1110	0110
7	1111	0111

4. המספר השלם הגדול במחשב

בשלבי חישוב שונים חשוב לדעת לעתים את הערך של המספר הגדול ביותר שנוכל להחזיק ב-m סיביות. בכל השיטות, הסיבית השמאלית ביותר של המלה מחזיקה את הסימן ולכן יש בה רק m-1 סיביות לייצוג הערך של המספר. נבחן כל שיטה בנפרד:

א. שיטת סימן וגודל

על ידי m-1 סיביות נוכל להחזיק מספר שערכו $2^{m-1}-1$ ולכן זהו המספר החיובי השלם הגדול ביותר במחשב. אם סיבית הסימן תהיה 1, נוכל להחזיק את אותו מספר עם סימן שלילי. מכאן, שהמספר הקטן ביותר שנוכל להחזיק הוא $-(2^{m-1}-1)$.

ב. שיטת המשלים ל-2

מכיון שמספר חיובי מוחזק בשיטה זו בדיוק כמו מספר חיובי בשיטה הקודמת, גם גודלו האפשרי לא ישתנה. בשיטה זו מייצגים מספר שלילי בדרך זו:

1	$2^{m-1} - \|X\|$

אם X שווה ל- -2^{m-1} נקבל $Z=|X|=2^{m-1}$ והתוכן של המלה יהיה:

0 0	1

זהו המספר הקטן ביותר שהמחשב יכול להחזיק. תוצאה זו סבירה, כי ככל שהערך $|X|$ יותר גדול, הערך של $2^{m-1}-|X|$ יהיה יותר קטן, ואין מספר יותר קטן מ-0. והערך המוחלט $|X|$ הגדול ביותר הוא 2^{m-1}, הערך הקטן ביותר של X הוא -2^{m-1}.

המספר השלילי הגדול ביותר שנוכל להחזיק במחשב (ז"א המספר הקטן ביותר מבחינת הערך המוחלט) הוא כאשר $|X|$ הוא הקטן ביותר. מצב זה קורה כאשר $2^{m-1}-|X|$ יהיה הגדול ביותר. הערך הגדול ביותר של הביטוי $2^{m-1}-|X|$ הוא: 1111111, שנקבל כאשר $|X|=1$. לכן, $X=-1$ מיוצג ע"י שורה של ספרות "אחד". מן הראוי לשים לב לכך שטווח המספרים השליליים שאפשר להחזיק לפי השיטה של המשלים ל-2 (בתחום $-1...-2^{m-1}$) גדול ב-1 מטווח המספרים השליליים שאפשר להחזיק לפי השיטה של סימן וערך (התחום $-1...-2^{m-1}-1$).

ג. שיטת עודף 2^{m-1}

ב-m סיביות נוכל לכתוב צירופים שונים של סיביות A, כאשר $0\le A\le 2^m-1$. מכיון שהמספר המיוצג N שווה ל: $A-2^{m-1}$, נובע שהגבולות של N הם:

$$-2^{m-1} \le N \le 2^m-1-2^{m-1} \quad \text{ז"א} \quad -2^{m-1} \le N \le 2^{m-1}-1$$

טווח המספרים זהה לזה שאפשר להחזיק לפי השיטה של משלים ל-2, כי הייצוג זהה חוץ מסיבית הסימן, אשר מוצגת בהיפוך.

4.1 ייצוג מספרים במחשבים שונים

נבחן את ייצוג המספרים בכמה מחשבים, שבהם מספרים שליליים מיוצגים בשיטת המשלים לשניים.

במחשב PDP-11, המלה היא בת 16 סיביות, כלומר m=16. מכאן שהמספרים השלמים X שהוא יכול להחזיק מוגבלים לתחום הזה:

$$-2^{15} = -32,768 \le X \le 2^{15} - 1 = 32,767$$

במחשבי יבמ בארכיטקטורה 370 המלה היא בת 32 סיביות, כלומר m=32. טווח המספרים הוא:

$$-2^{31} = -2,147,483,648 \le X \le 2^{31} - 1 = 2,147,483,647$$

במחשב VAX המצב דומה, אך יש שוני בכך שהייצוג של המספר הקטן ביותר (-2^{31}) איננו בשימוש ולכן המספר הקטן ביותר והמספר הגדול ביותר שווים בערכם המוחלט.

5. מספרים ממשיים בשיטת הנקודה הצפה

המספר העשרוני הגדול ביותר שנוכל לייצג בשני בתים בעזרת הקוד הדצימלי הוא 99 ובארבעה בתים המספר הוא 9999. ייצוג בינרי מאפשר להחזיק מספרים הרבה יותר גדולים ולכן הייצוג הפנימי של מספרים במחשב הוא בשיטה הבינרית.

למרות זאת, יש הגבלה בייצוג מספרים בינריים טהורים מכיון שהם מבטאים מספרים שלמים ולא שברים. בנוסף, גודל המספרים שנוכל להחזיק עדיין איננו מספיק לעבודות מדעיות ובמיוחד אם גודל המלה הוא 16 סיביות בלבד (כמו ב-PDP או ב-IBM-PC).

לבעיה זו יש פתרון בשיטה שונה של ייצוג מספרים. שיטה זו מאפשרת לייצג שברים וגם מספרים יותר גדולים מאשר ניתן לייצג בשיטות הקודמות, אשר טיפלו במספרים שלמים בלבד.

הייצוג של שברים באורך m ובבסיס b מקביל לייצוג של מספר שלם.

נתון: $F= 0.\, n_{-1}\, n_{-2} \dots n_{-m}$

הערך של F הוא: $\sum_{i=-1}^{-m} n_i\, b^i$ כאשר: $b^i = \frac{1}{b^{|i|}}$ עבור $i < 0$

בטבלה שלהלן מוצגים שברים בבסיסים שונים.

	0.1	0.01	0.001	0.0001
בינרי	1/2	1/4	1/8	1/16
אוקטלי	1/8	1/64	1/512	1/4096
הקסדצימלי	1/16	1/256	1/4096	1/65536

ייצוג של מספרים ממשיים במחשב הוא בשיטת נקודה צפה (Floating Point).
בשיטה זו מלת המחשב מחולקת לשלושה חלקים, כפי שנראה בתרשים:

סימן	מעריך	מנטיסה

הסבר:

א. ספרת סימן (Sign) 0 מיועדת למספר חיובי והספרה 1 - למספר שלילי.

ב. המעריך (Exponent) הוא החזקה בה צריכים להעלות בסיס b כדי לקבל את המספר המיוצג. מקדישים m סיביות למעריך ומציגים אותו בעודף 2^{m-1}, ז"א נוכל לייצג חזקות X כאשר $-2^{m-1} <= X <= 2^{m-1}-1$. הבסיס b שבו משתמשים נקבע בארכיטקטורה של המחשב.

ג. המנטיסה (Mantissa) היא חלק השבר של המספר ומניחים שיש משמאלה נקודה.

הערך של המספר שמייצגים הוא ערך המנטיסה כפול הבסיס מועלה לחזקת המעריך. לדוגמא, במחשב VAX ייצוג נקודה צפה נראה כך:

סיביות: 31 30...23 22........0

סימן	מעריך	מנטיסה

מקדישים 8 סיביות למעריך X, שהוא החזקה בה מעלים את הבסיס b=2. המעריך מוחסן בשיטת עודף 2^{8-1}, ז"א $-128<=X<=127$. לפיכך, הערך המינימלי שנוכל להכפיל בו את המנטיסה הוא 2^{-128} והערך הגדול להכפלה יהיה 2^{127}.

המנטיסה היא מספר בינרי רגיל. מניחים שהיא <u>מנורמלת</u> (Normalized), ז"א שיש בה 1 במקום הראשון מימין לנקודה.

בשיטת הנקודה הצפה אפשר לייצג כל מספר. לדוגמא נציג את המספר 5 ו-3/8 (חמש ושלוש שמיניות) ונסביר בשלבים:

א. $3/8= 1/4+1/8 = .011$
נובע מזה ש: 5 3/8 שווה ל: 101.011

ב. כדי לנרמל מספר זה, נכתוב אותו כך: $.101011*2^3$
הזזת הנקודה העשרונית שמאלה גורמת לשינוי חיובי בערך החזקה והזזת הנקודה העשרונית ימינה גורמת לשינוי שלילי בערך החזקה.

ג. הייצוג של 3 בעודף 128 הוא N+128, שערכו 131.

לדוגמא, נראה את הייצוג בנקודה צפה במחשב VAX. המנטיסה היא מספר מנורמל שמתחיל בספרה 1 משמאל. במחשב זה משמיטים את הספרה 1 שבמנטיסה משמאל, כי מניחים שהיא קיימת מבלי לכתוב אותה ומקבלים את תבנית הייצוג הבאה של המספר:

31	30 23	22 0
0	10000011	0101100..............00

על אף שיש רק 23 סיביות כדי להחזיק את המנטיסה, ההנחה שיש ספרה 1 משמאל נותנת אפקט כאילו יש 24 סיביות.

את המספר הקטן ביותר שנוכל להחזיק בנקודה צפה של ה-VAX נמצא בדרך זו: החזקה הקטנה ביותר היא 128- והמספר המנורמל הקטן ביותר הוא 0.1 (1/2). לכן המספר הקטן ביותר שאפשר לייצג היה צריך להיות:

$$2^{-128} * 1/2 = 2^{-129} = 1.47 * 10^{-39}$$

במחשב זה יש בעיה טכנית של הייצוג של המספר 0. גם אם המנטיסה מכילה רק אפסים, היא תחשב כייצוג של המספר 1/2 ולא של 0, כי הרי מניחים שהיא מתחילה בספרה 1. לכן החליטו להשתמש במעריך של 128- (ז"א שורה של אפסים בייצוג עודף 128) כדי לייצג את המספר 0. מכאן נובע שהמספר הקטן ביותר שניתן להחזיק במחשב הוא: $2^{-127}*1/2=2^{-128}=2.94*10^{-39}$.

יש גירסאות אחרות של פסקל, שמשתמשות במעריך 128- כדי לציין 0, רק אם המנטיסה מכילה אפסים בלבד, ולכן המספר הקטן ביותר הוא $1.47*10^{-39}$.

את המספר הגדול ביותר מחשבים כך: החזקה הגדולה ביותר שנוכל לייצג היא 127. המספר המנורמל הגדול ביותר שנוכל להחזיק הוא כמעט אחד (0.111...1 סה"כ 23 ספרות 1). הערך העשרוני של המספר המיוצג ע"י המעריך והמנטיסה האלה הוא: $2^{127}*1 = 1.7*10^{38}$

בעזרת ייצוג של נקודה צפה נפתרה הבעיה של גודל המספר וגם הבעיה של ייצוג שברים.

6. תרגילים

6.1 שאלות

(1) אילו יחידות קלט הוחלפו ע"י המסוף?

(2) האם יש הבדל בין נתונים לבין הוראות אשר בזיכרון?

(3) כתוב את המספרים הבינריים האלה כמספרים דצימליים:
1100110, 101001

(4) כתוב את המספרים ההקסדצימליים האלה כמספרים דצימליים:
AB3C, 13A, 4D, D4

(5) א. אם מחברים שני מספרים שליליים בייצוג של משלים לשניים, מתי תהיה גלישה?
כאשר אין גלישה, הראה שהסכום הוא אמנם הסכום הנכון עם הסימן הנכון.
ב. חזור על התרגיל עם שני מספרים חיוביים.

(6) נתונה מלת מחשב בת 8 סיביות.
כתוב את המספרים 27-, 61- בצורת: א. סימן וגודל
ב. משלים לשניים
ג. עודף 2^7

(7) כתוב את המספרים הבינריים האלה כמספרים דצימליים:
1000101, 110.101 (שים לב לחלוקה של המספר לחלק שלם ולשבר).

(8) הפוך את המספר 1011.00111 מבינרי לאוקטלי.

(9) הפוך את המספר AF.7BOE מהקסדצימלי לבינרי.

(10) איך מייצגים את המספר 1/32 בנקודה צפה במחשב VAX?

(11) לשם מה דרושים אוגרים במחשב?

נספח ב'

טורבו-פסקל

נכתב ע"י משה ראב

1. מבוא

טורבו-פסקל (Turbo Pascal) הינו ניב נוסף ומהדר לשפת הפסקל, אשר נכתב ע"י חברת Borland International Inc. שבקליפורניה, ארה"ב. מהדר זה פועל במיקרו-מחשבים ובמחשבים אישיים תחת מערכות ההפעלה CP/M-86, CP/M-80, MS-DOS, PC-DOS והוא הנפוץ ביותר במערכות אלו בגלל תכונותיו המיוחדות. הטורבו (קיצור של השם המלא) כולל בתוכו מהדר, עורך (Editor) המותאם לשפת הפסקל ותפריט לשליטה על אופציות המהדר.

הטורבו מאפשר כתיבה בפסקל סטנדרטי - תוך השמטות מסוימות, אשר תפורטנה בהמשך. הוא שונה ממהדרים אחרים בכך שהוא מאפשר שליטה מלאה במיקרו המחשב בצורה שקיימת בשפת האסמבלי (Assembly Language), ונותן כלים רבי עוצמה שקיימים רק בשפות עיליות מתקדמות. יתרונו הגדול במהירות ההידור הגבוהה וביעילות של התכנית בשפת מכונה שהוא יוצר. יתרון נוסף של הטורבו על פני מהדר פסקל אחר הוא בספריות של שגרות ה-Tools, שנוספו לו. הן מאפשרות לכתוב יישומים בתחומים שונים ומגוונים כמו ניהול מסדי נתונים, גרפיקה בהפרדה גבוהה, חלונות, יישומים מתמטיים ועוד.

להלן נתאר את המאפיינים העיקריים של הטורבו-פסקל ואת ההבדלים הבולטים בינו לבין הפסקל הסטנדרטי. אין כאן כוונה ללמד את הטורבו, אלא רק להציג את מגוון האפשרויות הגלומות בו ולאפשר למי שיודע פסקל ללמוד להשתמש בו.

2. מה אינו פועל בטורבו?

א. פקודות GET ו-PUT הושמטו מהשפה. לעומת זאת, הורחבו פקודות READ ו-WRITE כדי לבצע פעולות של הפקודות שהושמטו.

ב. הקצאת זיכרון דינמית: בגירסה 1.0 של הטורבו קיימות הפקודות MARK, NEW, RELEASE, בדומה לאלו שקיימות בגירסת UCSD של פסקל. הפקודה

DISPOSE נוספה בגירסה 2.0. שטחי זיכרון ששוחררו ע"י הפקודה DISPOSE יוכלו להגות מוקצים שוב בקריאות הבאות לפרוצדורה NEW (דבר זה קיים רק בכמה ממהדרי הפסקל).

ג. אין משתמשים בהצהרה PACKED בעבור מערך. הטורבו משתמש במערך דחוס בכל מקום שמתאפשר לו, מבלי שיש צורך להורות לו על כך במפורש ועל כן אין צורך בהצהרה ובפרוצדורות PACKED ו-UNPACKED.

ד. לא ניתן להעביר פונקציות ופרוצדורות כפרמטרים.

ה. תויות (Labels) שמוגדרות עבור פקודת GOTO יכולות להיות אלפא-נומריות ולא רק מספרים שלמים חיוביים. השימוש בהן זהה: יש לציין את התוית (או תויות) בהצהרה LABEL ובתחילת קטע התכנית שהתוית שייכת אליו כותבים את שמה ונקודתיים. לדוגמא,

```
LABEL  PART2;
BEGIN
 ::
  GOTO PART2;
   ::
 PART2: WRITELN;
        ::
```

ו. אי אפשר להשתמש בפקודת GOTO כדי לצאת מפרוצדורה, או מפונקציה (בפסקל סטנדרטי ניתן לעשות זאת במיגבלות מסוימות).

ז. במערכת הפעלה CP/M-80 בלבד, לא ניתן להעביר משתנה מקומי לפונקציה, או לפרוצדורה כפרמטר VAR, כאשר קוראים להן ברקורסיה.

3. סוגי נתונים וקבועים

סוגי הנתונים היסודיים הסטנדרטיים של פסקל הם: INTEGER (מספר שלם בתחום -32768..32767), REAL, BOOLEAN ו-CHAR.

בטורבו קיימים סוגי נתונים נוספים:

א. <u>BYTE</u> (בית) - מספר שלם בתחום 0 עד 255, אשר תופס בזיכרון בית אחד בלבד. ניתן להשתמש בו בביטויים אריתמטיים כמו ב-INTEGER.

ב. <u>STRING</u> (מחרוזת) - סוג נתון זה דומה ל-PACKED ARRAY OF CHAR. ההבדל הוא בכך שסוג זה הוא מחרוזת דינמית, שאורכה המקסימלי הוא 255 תווים. אם נצהיר VAR ST: STRING[30]
הצהרנו למעשה על משתנה מחרוזתי שאורכו המקסימלי 30 תווים. באיבר ST[0] נמצא את אורכה של המחרוזת ובשאר האיברים נמצאים תווים.
ניתן להתייחס לכל תו במחרוזת. לדוגמא: הביטוי STR[POS] יתן את התו הנמצא במקום POS שבמחרוזת STR.

ג. <u>קבוע מאופיין</u> (Typed Constant) - משתנה שערכו נקבע בהצהרת ה-CONST של התכנית וניתן לשנות את ערכו (אך יש להזהר בכך).
לדוגמא, אם כותבים: CONST KILOBYTE: INTEGER=1024;
ניתן להשתמש בערך המוצהר של KILOBYTE כאילו היה קבוע רגיל, אך גם ניתן לשנות את ערכו כמשתנה ולכתוב למשל KILOBYTE:=56;.
שימוש אחד של אופציה זו הוא לאתחל (לתת ערך התחלתי) משתנה, שהרי ערכו של משתנה אינו מוגדר (Undefined) עד אשר מציבים בו ערך כלשהו.

היתרון של הקבוע המאופיין בכך, שניתן להגדיר באמצעותו קבועים מסוגים מורכבים, כמו מערכים ורשומות.
יתרון נוסף הוא החיסכון בזיכרון. קבועים מאופיינים הם משתנים ולכן בזמן ההידור נרשמת בפקודה הכתובת שלהם בזיכרון, בעוד ששימוש בקבועים רגילים גורם להחלפת שם הקבוע בערכו בזמן ההידור. כאשר מופיע בתכנית פעמים רבות קבוע שאורכו גדול, יהיה חיסכון בשטח הזיכרון אם נשתמש בקבוע מאופיין במקום בקבוע רגיל.

הטורבו-פסקל מאפשר לפעול על סיביות של מספרים מהסוגים INTEGER ו-BYTE.
סדר הקדימויות של האופרטורים בחישוב הביטוי, הם:

א. NOT - דרגת קדימות כשל הסימן מינוס (ששייך למספר, להבדיל מפעולת חיסור שדומה בסדר הקדימות שלה לחיבור).
האופרטור NOT גורם להיפוך של סיביות המספר: סיביות "1" תהפוכנה לסיביות "0" ולהיפך. למשל, המספר 6 מיוצג כך במלה של 8 סיביות: 00000110. לאחר שנפעיל עליו NOT, צירוף הסיביות יהיה 11111001, שערכו למעשה 7- (בשיטת המשלים ל-2).
ב. AND, SHL (SHift Left), SHR (SHift Right) - דרגת קדימות כמו לכפל.
האופרטור AND גורם לביצוע פעולת AND של אלגברה בוליאנית בין כל שתי סיביות מתאימות בשני המספרים.
הפעולות SHL, ו-SHR מזיזות את הספרות הבינריות שמאלה או ימינה. למשל: 64 SHR 2, תהיה הזזה ימינה בשני מקומות של הסיביות בבית 0100000 ותתן את התוצאה 00010000 שערכה 16 (זוהי למעשה חלוקה ב-4).
ג. OR, XOR - דרגת קדימות כמו לחיבור.
OR ו-XOR אף הן פעולות של אלגברה בוליאנית כמו האופרטור AND.

בטורבו-פסקל ישנם סוגי קבועים, שאינם מוגדרים בשפת פסקל הסטנדרטית. מטרתם לאפשר קלט של תווים מיוחדים והדפסה שלהם, וגם לתת הוראות הפעלה למדפסת ולמחשב עצמו. סוגי הקבועים המשמשים למטרה זו:

א. קבועים הקסה-דצימליים (Hexadecimal Constants).
קבועים הקסה-דצימליים (בסיס 16) מציינים ע"י התו '$' משמאל לקבוע. לדוגמא, $A שווה בערכו ל-10 ו-$1B שווה בערכו ל-27 (16+11=27).
ב. תווי בקרה (Control Characters).
- קבועים שמתארים תו לפי מספרו בצופן ASCII, למשל: #10 שהוא למעשה "Line-Feed" ו-#$1B שהוא התו ה-27 ב-ASCII שמשמעותו "Escape".
- תווים הנקלטים כאשר מקישים אותם יחד עם מקש ה-CTRL. למשל, הצירוף ^G שווה ל-ציפצוף, והמשמעות של ^] היא Escape. ניתן לשרשר תווי בקרה ע"י כתיבתם בזה אחר זה כמחרוזת, כמו לדוגמא: #13#10^G#20.

4. פעולות במחרוזות

הפעולות שמותר לבצע במחרוזות הן:

א. פעולת השמה. `STR:= 'STRING'; STR2:= STR;`
ב. חיבור מחרוזות ע"י האופרטור '+'. `STR:= 'TURBO'+'^'+'PASCAL';`
ג. השוואה בין מחרוזות כאשר ההתייחסות היא לפי סדר המיון של התווים.
ד. פרוצדורות ופונקציות ספריה שונות.

הפרוצדורות המתאימות לפעולות במחרוזות:

DELETE(ST,POS,NUM) - מחיקת NUM תווים מהמחרוזת ST, החל ממקום POS;
INSERT(OBJ,TARGET,POS) - הכנסת המחרוזת TARGET לתוך המחרוזת OBJ מהמקום POS ואילך;
STR(VALUE,ST) - הפיכת מספר למחרוזת;
VAL(ST,VAR,CODE) - הפיכת מחרוזת למספר.

רשימת פונקציות לפעולות במחרוזות:

COPY(ST,POS,NUM) - הפונקציה מחזירה NUM תווים ממחרוזת ST, החל ממקום POS;
CONCAT(ST1,ST2,..,STN) - שירשור מחרוזות ST1, ST2, עד STN (ניתן להשתמש באופרטור '+' במקום פונקציה זו);
LENGTH(ST) - אורך מחרוזת;
POS(OBJ,TARGET) - המקום היחסי של המחרוזת OBJ במחרוזת TARGET.

5. הקצאת זיכרון

ניהול זיכרון דינמי מתאפשר בטורבו בעזרת הפונקציות והפרוצדורות הבאות: NEW, DISPOSE, MARK, RELEASE, GETMEM ו-MEMAVAIL. במהדורה 2.0 נוספו הפרוצדורות FREEMEM ו-MAXAVAIL. ארבע הראשונות קיימות גם בפסקל סטנדרטי.

נסקור בקצרה את התכונות של הפונקציות הנוספות בטורבו.

GETMEM(PVAR,N) הפרוצדורה משמשת להקצאת זיכרון עבור המצביע PVAR בגודל N בתים, ללא התייחסות לאיפיון של המשתנה או הזיכרון שעליו מצביע PVAR ולאורכו. הקצאת זיכרון זו משמשת בעיקר לניהול מחסניות.

MEMAVAIL הפונקציה מחזירה את גודל הזיכרון הנותר להקצאה דינמית (הגודל המקסימלי שניתן להקצות הוא 64KB). במערכות מחשב עם מלים של 16 סיביות, הערך המוחזר הוא גודל השטח במלים בנות 16 סיביות. במערכות של 8 סיביות הערך המוחזר הוא גודל השטח בבתים. האיפיון של הערך המוחזר הוא INTEGER; אם גודל הזכרון הפנוי עולה על 32767 בתים (גודלו של השלם המקסימלי), הערך המוחזר יהיה שלילי כך, שגודל הזיכרון הפנוי הוא 65767.0 פלוס הערך המוחזר.

FREEMEM(PVAR,N) פרוצדורה זו משחררת שטח שהוקצה בעזרת GETMEM. יש לשים לב לכך ש-N, מספר הבתים שמשחררים, יהיה זהה למספר שניתן בפקודת ה-GETMEM המקבילה.

MAXAVAIL הפונקציה מחזירה את גודל שטח הזיכרון המקסימלי שניתן להקצות מבין כל השטחים הפנויים. שטחי הזיכרון הפנויים אינם רצופים, אלא ניתנים בקטעים בהתאם להקצאה ולשחרור.

6. ניהול קבצים

כאשר עובדים עם קבצים בדיסק או בתקליטון אין צורך לציינם בסוגריים שבשורת הכותרת. יש לייחס לכל קובץ שמוגדר בתכנית את שמו החיצוני (כפי שמערכת ההפעלה מכירה אותו) בעזרת הפקודה הבאה:

```
ASSIGN (FILVAR, STR);
```

כאשר FILVAR הוא משתנה המוגדר בתכנית כקובץ ו-STR היא מחרוזת (או משתנה מחרוזתי) שמכילה את שם הקובץ החיצוני.

קבצי תמליל ניתן לפתוח לקריאה (RESET), לכתיבה (REWRITE) או להוספה. פעולת ההוספה לקובץ קיים אינה נמצאת בפסקל הסטנדרטי. בטורבו משתמשים למטרה זו בפקודה (APPEND(FILVAR.

גם לקבצים רגילים ניתן להוסיף רשומות, אך בדרך שונה. פותחים את הקובץ ע"י RESET, מציבים את מצביע הקובץ על סוף הקובץ ע"י הפרוצדורה SEEK וכותבים. כל כתיבה בקובץ תגרום להוספת רשומות.

```
RESET (FILVAR);
SEEK (FILVAR, FILESIZE(FILVAR));
```

הטורבו נותן פרוצדורות נוספות לפעולות בקבצים:

SEEK(FILVAR,N)	הצבה של מצביע הקובץ על הרשומה שמספרה N (מספר הרשומה הראשונה הוא אפס);
FLUSH(FILVAR)	הורקת המאגר (Buffer) של הקובץ לדיסק או לתקליטון. פעולת הקלט הבאה תתבצע ישירות מהדיסק, ולא מהמאגר שבזיכרון (נוהל זה חשוב בעיקר כאשר פותחים קובץ לקריאה ולכתיבה);
CLOSE(FILVAR)	סגירת קובץ. בטורבו חובה לסגור כל קובץ, משום שהפעולה של סגירת קבצים אינה מתבצעת באופן אוטומטי בסיום התכנית;
ERASE(FILVAR)	מחיקת קובץ;
RENAME(FILVAR,STR)	שינוי השם החיצוני של קובץ.

פונקציות נוספות מספקות מידע על מצב הקובץ (הן אינן פועלות על קבצי תמליל):

FILEPOS(FILVAR)	מחזירה את המספר היחסי של הרשומה בקובץ,
FILESIZE(FILVAR)	מחזירה את מספר הרשומות בקובץ.

מלבד קבצים רגילים וקבצי תמליל קיים גם קובץ חסר-איפיון ומטרתו לשמש לקלט/פלט ברמה נמוכה. קלט זה מתבצע ישירות בין הדיסק או התקליטון לבין המשתנים ובכך חוסך את השטח בזיכרון המוקצה לקובץ. ההצהרה על קובץ כזה היא:

```
VAR FILVAR: FILE;
```

הפרוצדורות READ, WRITE ו-FLUSH אסורות לשימוש בקובץ זה ובמקומן ניתנו הפרוצדורות הבאות:

```
BLOCKREAD (FILVAR, VAR, RECS);
BLOCKWRITE (FILVAR, VAR, RECS);
```

FILVAR הוא שם הקובץ, VAR הוא משתנה כלשהו שלתוכו קוראים, או שממנו כותבים את הרשומות ו-RECS הוא מספר הרשומות שיש לקרוא או לכתוב. היתרונות של השימוש בקובץ זה הם מהירות פעולות הקלט והפלט והחיסכון בשטח זיכרון.

שימוש מקובל בשיטה זו הוא לשם העתקה של קבצים.

הטורבו-פסקל מאפשר ניהול קבצים לגישה ישירה. חבילת התוכנה TOOL BOX מכילה כלים לניהול קבצים.

הטורבו מספק למתכנת מספר קבצים לוגיים מוגדרים מראש, אשר מציינים אמצעי קלט/פלט סטנדרטיים. ניתן להשתמש בהם מבלי שיהיה צריך להצהיר עליהם במפורש, לפתוח ולסגור אותם. אם נציין אותם כפרמטר הראשון בפקודות READ ו-WRITE, המחשב יידע להתייחס אליהם. לדוגמא,

```
WRITE (TRM, 'THIS IS AN EXAMPLE');
```

TRM הוא השם הלוגי של המסך. כאשר מציינים אותו כפרמטר הראשון בפקודה WRITE, הכתיבה תתבצע עליו, ולא על יחידת פלט אחרת.
קובץ לוגי אחר הוא המדפסת, ששמה הלוגי הוא LST. כדי להדפיס שורה במדפסת נכתוב לדוגמא, את הפקודה הבאה:

```
WRITE (LST, 'WRITING ON THE PRINTER');
```

קיימים מספר קבצים לוגיים נוספים המאפשרים ניצול מלא של כל אמצעי הקלט/פלט של המערכת, כולל תיקשורת חיצונית דרך מודם (אמצעי תקשורת בין מחשבים).

7. פעולות מסך וגרפיקה

שפת פסקל הסטנדרטית אינה מספקת פקודות או פונקציות ופרוצדורות לשליטה מלאה במסך. הטורבו מאפשר לדוגמא, באמצעות פרוצדורות ספריה שונות, להגיע עם הסמן לכל מיקום במסך, להוסיף שורה ולמחוק שורה, או קטעים מן המסך.

המסך מחולק לפי קואורדינטות X ו-Y: Y הוא מספר השורה ו-X הוא מספר הטור, כאשר הפינה השמאלית העליונה של המסך היא הנקודה (X=1,Y=1). כדי להגיע לכל מקום במסך משתמשים בפרוצדורה GOTOXY(XPOS,YPOS), אשר מציבה את הסמן בשורה YPOS ובעמודה XPOS.

להלן רשימת הפרוצדורות לשליטה במסך:

- CLRSCR - ניקוי המסך;
- CLEOL - מחיקת התוכן מהמקום בו נמצא הסמן ועד לסוף השורה;
- DELLINE - מחיקת השורה שבה נמצא הסמן והעלעת כל השורות שמתחתיה שורה אחת מעלה;
- INSLINE - הוספת שורה ריקה למטה מהמקום שבו נמצא הסמן, ע"י דחיפה של כל השורות שורה אחת למטה;
- NORMVIDEO, LOWVIDEO – שליטה על בהירות ההצגה ע"י הפרוצדורות NORMVIDEO (עוצמה רגילה) ו-LOWVIDEO (עוצמה חלשה).

במחשב IBM-PC אפשר לשלוט על הצבעים ועל הגרפיקה בהפרדה גבוהה, ע"י פרוצדורות סטנדרטיות. פרוצדורות נוספות מאפשרות להפיק צלילים בתדירות רצוייה, לקבוע את משך השהיית הצליל ולהפסיק אותו. לדוגמא:

```
SOUND (760);      (* יצירת צליל בתדירות 760 הרץ *)
DELAY (1000);     (* השהיית הצליל למשך שנייה אחת *)
NOSOUND;          (* הפסקת הצליל *)
```

חבילת תוכנה בשם TOOL GRAPHIX, כוללת פרוצדורות לשימוש בגרפיקה ובחלונות. תוכנה זו מכילה ספריה של פרוצדורות בפסקל, אשר מהוות למעשה פקודות רבות עוצמה ליצירת יישומים גרפיים. נציין מקצת מהפרוצדורות הקיימות:

- INITGRAPHIC - העברה של המסך למצב גרפי.
- DRAWPOINT(..) - סימון נקודה על המסך.
- DRAWLINE(..) - שרטוט קו ישר בין שתי נקודות.
- DRAWCIRCLE(..) - שרטוט מעגל או אליפסה (בהתאם לפרמטרים).
- HARDCOPY(..) - הדפסה של המסך (שנמצא במצב גרפי) במדפסת.
- DRAWBORDER - שרטוט קו על גבולות המסך.
- LEAVEGRAPHIC - החזרת המסך ממצב גרפי למצב רגיל.

8. מבנה התכנית

הצהרות

בפסקל סטנדרטי מבנה התכנית קבוע. בתחילת התכנית מופיעות כל ההצהרות בסדר שאינו ניתן לשינוי:

```
PROGRAM XXXX;
  LABEL ...;
  CONST ...;
  TYPE ...;
  VAR ...;
  PROCEDURE ...;
   : : :
    BEGIN
     : :
```

הטורבו מאפשר לשנות את סדר ההצהרות ולא מגביל את מספר הפעמים שמופיעה כל הצהרה. באופן זה יכולה למשל, הצהרת ה-CONST להופיע פעמיים (או יותר) בתחילת התכנית, או בפרוצדורות, כפי שמודגם למטה. האפשרות הזו דורשת תשומת לב רבה שלא להשתמש בקבוע או במשתנה שטרם הוגדר ולא להגדיר <u>מזהים</u> (Identifiers) יותר מפעם אחת.

```
PROGRAM xxx;
  CONST VECSIZE=20; LENGTH=18;
  VAR S: REAL;
  TYPE TABLE= ARRAY [1..VECSIZE] OF REAL;
  VAR MEDIDOT: TABLE; NAME: STRING[LENGTH];
  CONST MDIDAMAX=67.0;
  BEGIN
   : :
```

בטורבו מקובל השימוש בחבילות של שגרות ה-Tools, שאינן אלא קבצים שמכילים פרוצדורות, פונקציות והגדרות של משתנים, קבועים ואיפיונים. כל קובץ כזה מספק למתכנת כלים מוכנים מראש לבניית יישום בנושא מסוים ואפשר לכלול אותו

בתכנית ע"י פקודת INCLUDE. הסרת המגבלה של סדר ההצהרות מאפשרת להגדיר משתנים גלובליים גם בין השגרות, בנוסף להצהרות על משתנים, קבועים ואיפיונים אשר ניתנות בקבצים ומשמשות בתכנית בה הם מוכלים.

שגרות Tools כאלו קיימות בנושאים רבים ומגוונים: ליישום שדורש ניהול קבצים נוכל להשתמש בחבילת שגרות ה-Tool Box, שגרות אחרות מיועדות לתמיכה ביישומים מתמטיים, לתמיכה בגרפיקה בהפרדה גבוהה ועוד.

<u>מגבלות הזיכרון וקבצי OVERLAY</u>

בטורבו קיימת מגבלה של אורך השורה ושל גודל התכנית. אורך השורה מוגבל ל-127 תווים כפי שנקבע ע"י העורך, וגודל התכנית אחרי ההידור איננו יכול לעלות על 64KB. הגבלת הגודל נובעת מאופי החומרה במערכות מיקרו-מחשבים שבהן התוכנה פועלת. בדרך כלל לא כותבים תכניות בגודל זה, אך אם בכל זאת צריך לכתוב תכנית ארוכה, ניתן לעקוף את המגבלה ע"י שירשור תכניות, או ע"י יצירת קבצי OVERLAY.

שירשור תכניות היא האפשרות לעבור בתוך כדי ביצוע תכנית אחת, להרצת תכנית אחרת. אם היישום שכותבים ארוך מידי יש אפשרות לפצלו לשתי תכניות ולקשר ביניהן.

קבצי OVERLAY הם קבצים המכילים פרוצדורות ופונקציות. כאשר גודל התכנית חורג מ-64KB, מגדירים קבוצה של פרוצדורות ופונקציות כ-OVERLAY, ע"י כתיבת המלה OVERLAY בשורת ההגדרה שלהן. לדוגמא:

```
OVERLAY PROCEDURE XXX(PAR1);
OVERLAY FUNCTION YYYY(PAR1,PAR2);
```

כל הפרוצדורות והפונקציות שהוגדרו כך יהיו בקובץ נפרד מהתכנית הראשית, אשר בה יישמר מקום ריק המתאים לגודל הפרוצדורה הארוכה ביותר שבקובץ ה-OVERLAY. נקרא לשטח הזה שטח OVERLAY. כאשר התכנית הראשית מגיעה לשלב שבו יש קריאה לפרוצדורה שנמצאת בקובץ ה-OVERLAY, היא קוראת לה מהקובץ אל השטח השמור בה למטרה זו ואז עוברת לביצוע שלה. השימוש בקבצי OVERLAY מאיט את ההרצה של התכנית משום שכל קריאה לפרוצדורה, או פונקציה, שאינה נמצאת בשטח ה-OVERLAY דורשת גישה ליחידת החסנה חיצונית.

<u>9. שילוב שגרות בשפת מכונה</u>

כבר צויינה בתחילת הפרק יכולתו של הטורבו-פסקל לבצע פעולות שבד"כ אינן בתחום השליטה של שפות עיליות. בטורבו אפשר להשתמש בשגרות בשפת מכונה ובכך לחרוג מהמגבלות שמוטלות על השפה מהיותה שפה עילית.

להלן נפרט כמה מהאפשרויות האלו. לפני כן נקדים ונאמר שהטורבו פועל במספר מערכות הפעלה, שפועלות על מיקרו-מעבדים שונים: 8086 ו-8088, Z80 ו-Z80/A. מכיון שמיפוי הזיכרון (מבנה החומרה) שונה במעבדים השונים נמצא גם .הבדלים קלים, אך לא משמעותיים, במבנה הפקודות.

הדוגמאות שלהלן שייכות למיקרו-מעבדים מסוג 8086 ו-8088 (מערכות הפעלה

MS-DOS, PC-DOS, ו-CP/M-86), אשר נמצאים במחשב IBM-PC ותואמיו. במערכות אלו כל כתובת מתוארת בעזרת שני פרמטרים: מספר קטע הזיכרון (Segment) שבו נמצאת הכתובת, והמיקום היחסי שלה (Offset) בתוך אותו קטע. גודלו של כל קטע בזיכרון הוא 64KB.

הדרך לכתיבת שגרות בשפת מכונה היא ע"י הפרוצדורה INLINE. הפרמטר של הפרוצדורה תהיה השגרה בשפת מכונה, אשר תוצג בשיטה הקסה-דצימלית. כל 8 סיביות, שהן שתי ספרות הקסה-דצימליות, תופרדנה ע"י קו נטוי ('/'). לדוגמא:

```
BEGIN
  WRITELN ('THE NEXT LINE IS A MACHINE LANGUAGE ROUTINE');
  INLINE ($50/$53/$51/$52/$57/$56/$06/$FB);
          (* רוטינה זו חסרת משמעות ולכן אין לנסות להריצה ! *)
    : : :
```

השגרה בשפת מכונה אינה קריאה משום שהיא כתובה בקודים הקסה-דצימליים של פקודות שפת המכונה. לכן, כדי לשלב שגרות של שפת מכונה ע"י פרוצדורת INLINE, יש לכתוב אותן תחילה באסמבלי ולתרגם לשפת מכונה, או להשתמש בשגרה מוכנה.

בטורבו ישנה אפשרות לפנות לכל כתובת בזיכרון המחשב. ניתן לדעת את מיקומו של כל משתנה ולהציב בכל מצביע כל ערך כך שיצביע על המקום הרצוי בזיכרון. לדוגמא, קיימות פונקציות ספריה המאפשרות לקבל מידע על מיקומו של משתנה בזיכרון המחשב, או לקבוע מראש את כתובתו של כל משתנה:

ADDR(VAR)	פונקציה המחזירה ערך בגודל 32 סיביות, שהוא כתובת בזיכרון של המשתנה VAR. ערך זה ניתן להציב במצביע כדי שיורה על אותו משתנה.
SEG(VAR)	הפונקציה מחזירה את מספר הקטע בזיכרון שבו נמצא המשתנה VAR.
OFS(VAR)	הפונקציה מחזירה את המיקום היחסי של המשתנה בקטע.

שימוש נוסף בשגרות בשפת מכונה הוא לשם גישה ל"יציאות" (Ports) של המחשב, לשם הפעלה וניצול התכונות שלהן.

10. הוראות למהדר

מספר מאפיינים של המהדר ניתנים לשליטה דרך התכנית באמצעות הוראות הנקראות Compiler Directives. הן מאפשרות להכליל או להשמיט פעולות סטנדרטיות אשר מסופקות ע"י המהדר בזמן ההידור של התכנית. בתנאים מסוימים התכניתן יכול לקבוע שפעולות מסוימות דרושות, או שאינן דרושות לכל התכנית או לחלקים ממנה. בכך הוא יכול לייעל אותה ולשפר את זמן הביצוע שלה. למשל, ניתן לשלוט על שיבוץ או השמטה של בדיקת שגיאות (של זמן הרצה) בתכנית. לאחר שמנפים מהתכנית את כל השגיאות אפשר להשמיט את הבדיקות כדי שהתכנית "תרוץ" מהר יותר ותהיה יותר קצרה.

את ההוראות למהדר נותנים בתוכנית באמצעות הערות, אשר התו הראשון בהן הוא '$' והתו השני הוא אות שמתארת את ההוראה. אם זוהי פעולה שניתן להפעילה או להפסיקה, יש לכתוב לאחר האות את הסימן '+' (להפעיל), או '-' (לא להפעיל).

לדוגמא, כתיבת ההערות בשתי דרכים שונות:

```
{$C+ }              או      (*$C+ *)
{$I INCL.FIL }      או      (*$I INCL.FIL *)
{$B-,R+,V- }        או      (*$B-,R+,V- *)
```

בשפת פסקל הסטנדרטית וגם בטורבו ניתן לכתוב את ההערות בין סוגרים גדולים, או בין סוגרים עגולים שבצידם הפנימי יש כוכבית. את ההערות אפשר לכתוב בכל מקום בתכנית שמופיע בו תו הפרדה (רווח, פסיק או נקודה-פסיק) ולכן גם ההוראות למהדר יכולות להיכתב באותם מקומות. בדרך זו ניתן לשנות את מאפייני ההידור בכל חלק של התכנית.

להלן נתאר כמה מן ההוראות למהדר:

א. בדיקות קלט/פלט מציינים ע"י קוד הפעולה I. כאשר האופציה מופעלת, כלומר {$I+}, כוללים בתכנית בדיקות הקשורות בביצוע של פעולות קלט/פלט שיגרמו להפסקת ההרצה במקרה של שגיאה. כברירת מחדל, אופציה זו מופעלת. כאשר אין מפעילים אותה, על המתכנת לערוך את הבדיקות הדרושות. הציון של שגיאה בביצוע של פעולות קלט/פלט ניתן באמצעות הפונקציה IORESULT, אשר מחזירה ערך שונה מאפס אם היתה שגיאה. לדוגמא,

```
{$I-} RESET(FILVAR) {$I+};          { ניסיון לפתיחת קובץ לקריאה }
OK:= (IORESULT=0);
  { OK - משתנה בוליאני שיכיל FALSE אם היתה שגיאה בפתיחת הקובץ }

IF NOT OK THEN WRITELN ('ERROR IN OPENING FILE !!!');
```

ב. מפעיל התכנית יכול לתת הוראה להפסיק את ההרצה של התכנית בכל שלב ע"י הקשת Ctrl-C. ההרשאה לכך ניתנת ע"י קוד הפעולה U. ברירת המחדל היא {$U-}, כלומר שאין אפשרות להפסיק הרצה. הפעלת האפשרות הזו בתכנית, דהיינו כתיבת {$U+}, גורמת לירידה במהירות הביצוע שלה.

ג. קיימות הוראות שאפשר לתת למהדר פעם אחת בלבד בתכנית ולאחר שניתנו, לא ניתן לשוב ולשנותן במהלך אותה תכנית. דוגמא לכך היא ההוראה לאפשר קליטת פסק (Interrupt) כלשהו בזמן פעולת התכנית:

- הקשה של Ctrl-C לשם הפסקת הרצה,
- הקשה של Ctrl-S כדי לעצור במהלך התצוגה על המסך.
- הקשה של Ctrl-Q כדי להמשיך את התצוגה על המסך, לאחר הפסקה.

ההוראה ניתנת ע"י קוד הפעולה C כך, ש-{$C+} מאפשר לקלוט הפרעות, ו-{$C-} איננו מאפשר זאת. ברירת המחדל היא {$C+}. אם מרגישים שהפלט איטי מידיי כתוצאה מכך, יש להורות {$C-}.

ד. שילוב של קובץ חיצוני בתכנית נעשה באמצעות הוראה למהדר. ההוראה גורמת לכך, שבזמן ההידור יתוסף לתכנית הקובץ שנמצא בהוראת ה-INCLUDE, במקום שבו כתובה ההוראה ויהיה חלק בלתי נפרד ממנה. היתרון שבכך הוא באפשרות לכתוב תכניות שאורכן גדול ממה שהעורך (Editor) מתיר. הדבר תורם גם לבהירות התכנית ומאפשר להשתמש בתכניות (בעיקר בשגרות) שנכתבו ונופו ע"י מישהו אחר.
התחביר של הוראה זו שונה במעט משאר ההוראות. קוד ההוראה הוא האות I, אך במקום הסימנים '+' או '-' אחריה, משאירים רווח ולאחריו כותבים את שם הקובץ שרוצים לכלול בתכנית. לדוגמא: {$I INCLUDE.FIL}
אם לא נציין בשם הקובץ את הסוג שלו (המלה שמופיעה אחרי הנקודה בשם הקובץ), ברירת המחדל היא שזהו קובץ פסקל. ההוראה {$I PROGI} למשל, תובן כמו ההוראה {$I PROGI.PAS}.

ביבליוגרפיה

<u>ספרים בעברית</u>

א. טאובר, דני ורוני: "פסקל", אקדמון, תשמ"ד (1984).

ב. "תכנות מבני בשפת פסקל" (מעובד), ביה"ס לטכנולוגיה של האוניברסיטה הפתוחה, תשמ"ו (1986).

<u>ספרים בסיסיים</u>

1. Atkinson, L. [1980]: "Pascal Programming", John Wiley & Sons LTD, New York.
2. Cherry, G. [1980]: "Pascal Programming Structures: An Introduction to Systematic Programming", Reston Publishing House, Reston, VA.
3. Findlay; W., Watt, D. [1978]: "Pascal: An Introduction to Methodical Programming", Computer Science Press, Potomac, MD.
4. Holt, R.C., Hume, J. [1980]: "Programming Standard Pascal", Reston Publishing House, Reston, VA.
5. Mazlack, L.: "Structured Problem Solving with Pascal", Holt, Rinehart and Winston, New York.
6. Rohl, J.S. [1980]: "Programming VIA Pascal", Cambridge University Press, Cambridge.
7. Schneider, G., Weingart, S., Perlman, D. [1978]: "Programming and Problem Solving with Pascal", Wiley, New York.
8. Tremblay, J., Bunt, R., Opseth, L. [1980]: "Structured Pascal", McGraw-Hill Inc., New York.
9. Tucker, A. [1983]: "Introduction to Programming with ESP Pascal", Holt, Rinehart and Winston, New York.
10. Welsh, J., Elder, J. [1980]: "Introduction to Pascal", Prentice-Hall, Englewood Cliffs, New Jersey.
11. Wilson, I.P., Addyman, A.M. [1978]: "A Practical Introduction to Pascal", MacMillan Press, London.

ספרים למתקדמים

1. Jones, W. [1982]: "Programming Concepts, A Second Course (With Examples in Pascal)", Prentice-Hall, Englewood Cliffs, New Jersey.
2. Lecarme, O., Nebut, J. [1984]: "Pascal for Programmers", McGraw-Hill Inc., New York.
3. Schneider, G., Bruell, S. [1981]: "Advanced Programming and Problem Solving with Pascal", John Wiley & Sons, New York.
4. Tenenbaum, A., Augenstein, M. [1981]: "Data Structures Using Pascal", Prentice-Hall, Englewood Cliffs, New Jersey.

טורבו-פסקל

1. (-) [1985]: "Turbo Pascal - The Ultimate Pascal Development Environment", Borland International Inc., Cal.

מבוא לעיבוד נתונים ומבנה המחשב

א. האן י', עמיהוד י': "המחשב האלקטרוני - עיבוד נתונים ותכנות בייסיק", הוד-עמי, רמת-גן, 1985.

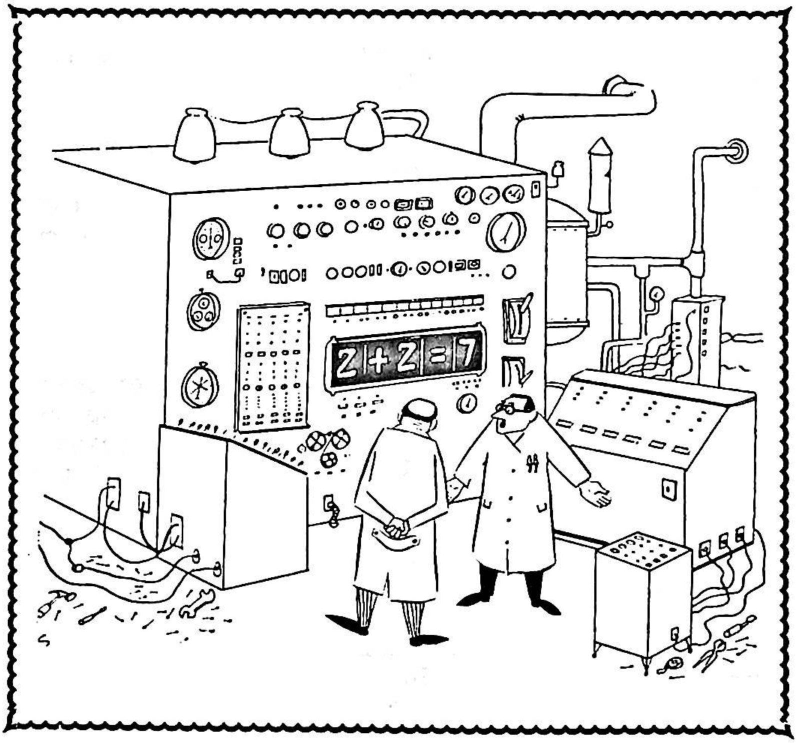
ב. עמיהוד י': "תכנון ותכנות יישומי מחשבים", הוד-עמי, רמת-גן, 1975.

ג. הייפרמן ר': "ארגון נתונים וניהול קבצים", הוד-עמי, רמת-גן, 1982.

1. Gear, C. W. [1974]: "Computer Organization and Programming", McGraw-Hill, New York.
2. Stone H. and Siewiorek D. [1975]: "Introduction to Computer Organization and Data Structures: PDP-11 Edition", McGraw-Hill, New York.

אינדקס ומילון מונחים

מילון המונחים הקצר מגיש לקורא את המונחים העיקריים שהוא נזקק להם כאשר הוא לומד את שפת פסקל.
במילון משולב גם אינדקס, אשר נותן את מראי המקומות העיקריים של הדיון בנושא או במונח.



2+2=7

ספרים נוספים בהוצאת הוד-עמי:

1. המחשב האלקטרוני - עיבוד נתונים ותכנות בייסיק, 1985
   - י' האן ו-י' עמיהוד
2. תכנון ותכנות יישומי מחשבים, 1975
   - י' עמיהוד
3. ארגון נתונים וניהול קבצים, 1982
   - ר' הייפרמן
4. בסיסי נתונים - עקרונות, מודלים ויישומים, 1984
   - ר' הייפרמן
5. מערכות תקשורת נתונים, 1981
   - א' סקופ
6. יסודות בניתוח ותכנון מערכות מידע, 1980
   - י' עמיהוד ו-י' קורפל
7. מיקרו-מחשבים וה-6800, חומרה ותוכנה, 1983
   - א' הראל

הספר "המחשב האלקטרוני" אושר ע"י משרד החינוך כספר לימוד. שאר הספרים אושרו כספרי עזר והעשרה.

* איך מיישמים את עקרונות **התכנות המבני** בשפת פסקל?
* איך מתכנתים תנאים בוליאניים, לולאות והסתעפויות, פונקציות ופרוצדורות?
* איך מעבדים נתונים במערכים ובקבצים?
* מה הם מבני נתונים סטטיים ומה הם מבנים דינמיים?
* מה הם קבצי תמליל ואיך כותבים עורך תמליל (Editor)?
* מה הן הפרוצדורות לשליטה במסוף?
* מהי רקורסיה ואיך היא מיושמת בפסקל?
* מהו **טורבו־פסקל** ומה הם ההבדלים בינו לבין פסקל?

פסקל — שפה לתכנות מבני

מציג בפני הקורא והלומד את שפת פסקל ואת יסודות התכנות המבני. הוא מדריך את הקורא בהדרגה ומאפשר לו ללמוד את עקרונות התכנות בשפת פסקל באמצעות דוגמאות תכנות רבות, אשר מלוות בהסברים, תרשימי זרימה ותרשימי הסבר שונים. בסוף כל פרק ניתנו שאלות לחזרה ותרגילי תכנות רבים.

פסקל — שפה לתכנות מבני מאפשר ללומד השקדן ללמוד באופן עצמאי. הוא מקיף את כל נושאי התכנות בשפת פסקל ברמה תיכונית ואוניברסיטאית ועל כן מתאים לשלב הלימוד המתקדם. הלומד יוכל להיעזר בספרות הענפה המפורטת בביבליוגרפיה, ובמילון המונחים והאינדקס שבסוף הספר. הנספח על טורבו־פסקל והביבליוגרפיה, יאפשרו להכיר ולהשתמש גם בניב זה של השפה.

הספר ערוך לפי הדרישות של תכנית הלימודים החדשה של משרד החינוך והתרבות עבור מגמות מחשבים־ענ״א בבתי הספר העל־יסודיים ובמכללות להנדסאים. הוא מותאם לדרישות מרכז ענ״א במכון לפריון העבודה והייצור, המכון להכשרה טכנולוגית (מה״ט) ותכניות הלימודים באוניברסיטאות ובמכללות למחשבים.

הספר נועד גם כספר עזר לכל המשתמש בשפת פסקל כשפת תכנות ושירות בתחומי עיסוקו.

המחבר:
פרופ׳ אבא אנגלברג הינו בעל תואר דוקטור בחקר ביצועים מאוניברסיטת ניו־יורק (NYU) משנת 1969. לאחר הוראה במשך שלוש שנים בפוליטכניקום של ניו־יורק (Politechnic Institute of NY) עלה ארצה, ומאז הוא מנהל את המחלקה למחשבים ומרצה בביה״ס הגבוה לטכנולוגיה בירושלים.
פרופ׳ אנגלברג פרסם מאמרים בכתבי עת מקצועיים בנושאים של חקר ביצועים ומחשבים וכתב ספר בסטטיסטיקה.